Creating Functions and Subroutines in CPQ Transcript

Functions and Subroutines provide useful efficiency. They enable you to create reusable sections of code, and quickly reference and execute that code in multiple places.

At the end of this course, you'll be able to describe the differences and purposes of functions and subroutines.

leverage scopes to manage the accessibility of your Functions and Subroutines

And Implement functions and subroutines for more efficient Snap code,

Select the right arrow icon to get started.

Objectives

- Functions and Subroutines Overview
- The Scope for Function or Subroutine
- Using Functions
- Using Subroutines

Functions and Subroutines Overview

Functions and Subroutines are reusable sections of Snap code you can write once, and then use many times from many places. Use functions and subroutines to simplify your logic into modular units that are easy to update and extend.

Declaring, or writing, code as a Function or a Subroutine creates a self-contained set of snap blocks. Using that function or subroutine is a simple, one-line Snap block

If you need to update your logic, you only need to change the declaration once, instead of having to update every individual place that you would use that code.

Functions and Subroutines are very similar. First, they both can have data passed into them, called arguments.

This data is used by the function or subroutine to perform calculations.

Functions and Subroutines can also make changes directly to the configurator or scene they are written in. Some examples of these changes, called side-effects, are changing the maximum value of a slider on a field, or making a page invisible.

But Functions and Subroutines do have a difference. Functions can pass data back out as a result. Subroutines can't pass any data back out. Subroutines only have side-effects.

For example, maybe your configurator needs to perform the same mathematical calculation in many places, such as calculating volume based on Height, Width, and Depth. Those 3 arguments can be passed in to a function, used in calculations, and then the result would be returned back out.

You could then do anything you want with that returned value, such as store it in a field, or use it for other purposes.

You could do the same calculation with a Subroutine, but this limits your flexibility. Subroutines return no result, so that assignment of the volume field would have to happen within the subroutine as a side effect. You couldn't do anything else with that value.

Here's a best practice to follow.

If you want a result back from your code that you can do anything with, use a function.

If you'd rather have the code act on that result and make a change to the configurator directly instead of returning the result to you, then use a subroutine.

The function in this example gives us more flexibility in case we wanted to do something else with the cube volume.

Select the right arrow icon for the first knowledge check.

The Scope for Functions and Subroutines

You can control where your functions and subroutines can be used. Maybe one function is only useful within a value rule of a configurator, but another function is useful across your whole organization. You control this visibility, or Scope, by where you declare your code.

A function or subroutine can be declared in one of three places. From smallest scope to largest, these are

In a rule, in a global rule, or as a safe function.

Let's start small. The narrowest scope for your function or subroutine would be to declare it in a rule in your configurator. When declared in a rule, such as a value rule, that logic can be used only in that rule or any rules listed in the same folder beneath it.

No other rules in that configurator, or any other configurators or scenes, can use this logic.

That's one place you can declare your logic. But what if you want your logic to be used anywhere throughout your configurator? Then broaden the scope of your logic by declaring it in a special rule, called a global rule.

Global rules are not part of the rule cycle. They help you build a reference library of code. Declare a function or subroutine there, and it can be used anywhere in your configurator.

But what if you've written code that is so useful, it could be used by other configurators? Or even by other system-wide features, like scenes or workflows?

What if you need to use server-side resources in your function, like sending email notifications, querying integration databases or using web services?

Or what if your code contains sensitive, proprietary logic you want to carefully control?

All three of these needs can be met with the third place your logic can be declared: a Safe function.

Safe functions are resources, like tables or media. Like any other resource, their scope is company-wide. They are independent units of code to be used by any configurator, quote, scene or workflow.

The code from a safe function remains in the cloud and is never sent to the user's browser. This means that it's safe from reverse engineering because your customer can never see the code within it. Keep in mind that safe

functions are slower than other logic. Results are often returned in a few seconds, rather than milliseconds, because information must travel from your customer's device up to the cloud and back again.

Learn more about safe function specifics in the documentation.

Given these three places to declare your logic, where should you start? A best practice is to create your function or subroutine in the configurator first, in a Global Rule. Later, if you discover that scope is too broad or too narrow, you can cut and paste your Snap blocks into another location.

Select the right arrow icon to continue to the next topic.

Using Functions

Let's build our first function. If you're building a function or subroutine in a rules folder, we recommend making a declarations work space. Place it at the top of the Rules order. If a function or subroutine is called before it is declared, the system produces an error.

Find the Function and Subroutine blocks in the definitions category.

For an example, we will build a function that returns the volume of a cube in our configurator.

Then we will show you a subroutine that calculates the volume of that same cube and updates a volume field directly.

Drag the one you want into your workspace.

Follow along as we build the function first.

These blocks attach to the definitions portion of the start block.

Name the function, and set the value type for it to return. In this case we'll name it CalculateVolume, and it returns a number.

For us to calculate the volume, we're going to need the values of height, width, and depth from the cube. In order to receive those values, our function needs three parameters added. Think of these parameters as slots where the arguments or values of height, width, and depth can be entered in to.

Select the plus icon to add those.

Now set the data type for each parameter and name it. In this case, all three are numbers. Label them x, y, and z.

We're designing a function, so we'll need to end it with a return block. Select the block to open up the buddy blocks panel. Take return and attach it.

Before we return a value, we need to calculate it. Insert the Math Operation Block, and set up the calculation for X times Y times Z.

Select the Get block from the Variables category, and set it into the math operation. Notice that when we open the dropdown, we see x, y, and z as options. These are referencing the parameters we just set up.

Finish setting up the equation. Now, when we call the function, it will return the value of x times y times z.

Let's take a look at how to call the function.

Open the workspace you want to call it in. in this case, we want to use this function to set the Volume field.

Now, lets call the function. In the Functions category, select Call function, and add it to the Set Field block.

Select the CalculateVolume function.

Notice the x, y, and z arguments we created earlier. Whatever value we place in these spots will be used in our function as x, y, and z. In this case, we want the height, width, and depth.

Now, this call function block is performing all of the code we created in the CalculateVolume function from our Declarations workspace.

Let's save and run our configurator to make sure it's working correctly.

As we adjust the sliders, the Volume field updates appropriately. Our function works as intended!

Select the right arrow icon to take a knowledge check.

Using Subroutines

Now, let's try to make this happen another way… Using a subroutine instead of a function.

Remember that a subroutine cannot return any value, so updating the Value field needs to be part of the subroutines code.

Name the subroutine… let's go with SetVolume, and add the same 3 parameters as before.

Because a subroutine does not return a value, we need to set the Volume Field inside the subroutine code.

So, let's add the Set Field Value Block with the same math equation we used in the function.

With our subroutine set up, now we have to call it. Let's go to our Set Volume workspace.

In the functions category, pick the Call subroutine block and attach it.

Select the subroutine and enter the height, width, and depth fields for the x, y, and z parameters.

Now when this code runs, it will perform the SetVolume subroutine's code using these values. Let's save and run.

As we adjust the sliders, the Volume field updates appropriately. Our subroutine works as intended!

Select the right arrow Icon for a quick knowledge check.

Recap

Functions and Subroutines provide useful efficiency. They enable you to create sections of code, and quickly reference and execute that code in multiple places.

You should now be able to describe the differences and purposes of functions and subroutines.

leverage scopes to manage the accessibility of your Functions and Subroutines,

And Implement functions and subroutines for more efficient Snap code.

About Epicor

Epicor Software Corporation drives business growth. We provide flexible, industry-specific software that is designed around the needs of our manufacturing, distribution, retail, and service industry customers. More than 40 years of experience with our customers' unique business processes and operational requirements is built into every solution—in the cloud, hosted, or on premises. With a deep understanding of your industry, Epicor solutions spur growth while managing complexity. The result is powerful solutions that free your resources so you can grow your business. For more information, connect with Epicor or visit www.epicor.com.

EPICOR

Corporate Office
804 Las Cimas Parkway
Austin, TX 78746
USA
Toll Free: +1.888.448.2636
Direct: +1.512.328.2300
Fax: +1.512.278.5590

Latin America and Caribbean
Blvd. Antonio L. Rodriguez #1882 Int. 104
Plaza Central, Col. Santa Maria
Monterrey, Nuevo Leon, CP 64650
Mexico
Phone: +52.81.1551.7100
Fax: +52.81.1551.7117

Europe, Middle East and Africa
No. 1 The Arena
Downshire Way
Bracknell, Berkshire RG12 1PU
United Kingdom
Phone: +44.1344.468468
Fax: +44.1344.468010

Asia
238A Thomson Road #23-06
Novena Square Tower A
Singapore 307684
Singapore
Phone: +65.6333.8121
Fax: +65.6333.8131

Australia and New Zealand
Suite 2 Level 8,
100 Pacific Highway
North Sydney, NSW 2060
Australia
Phone: +61.2.9927.6200
Fax: +61.2.9927.6298