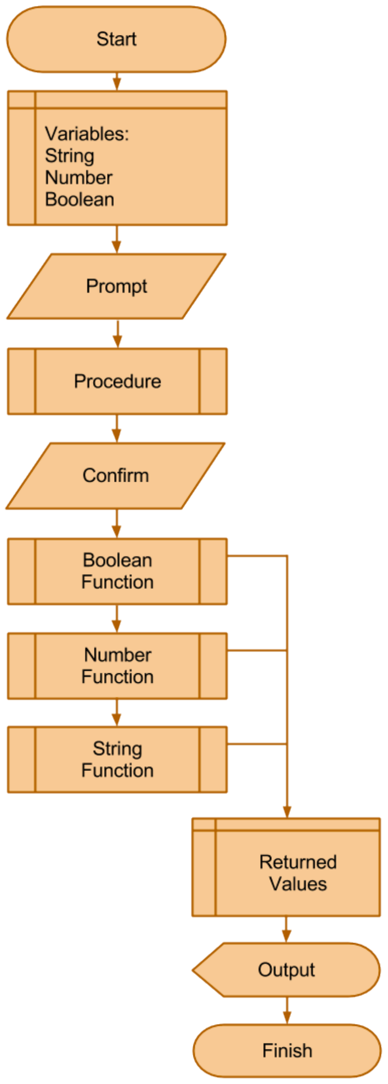
Flowchart Symbols and What They Mean

The Project 2 flowchart can seem a bit daunting, but if you break it down into its pieces and parts, you'll see that it's not as scary as it looks. To begin, let's take a look at the flow of the main code.



We know that this part of the flowchart is the main code because it contains the **Start** and **Finish** support details in the **Terminator** symbols. Remember that these indicate where our code will begin and where it will end.

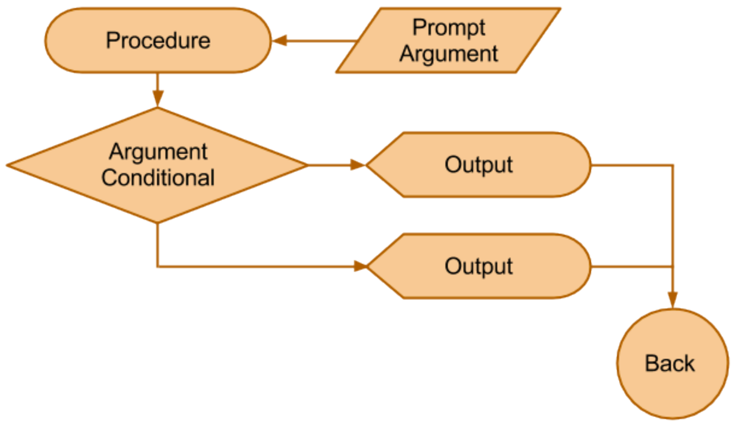
The next symbol of the flowchart indicates **Internal Storage**, which is the technical term for stored values. This symbol also contains **support details** that tell us what should be happening in our code at this time. In this case, we need to declare and define our string variable, number variable, and Boolean variable.

The next symbol in the flow is the **Data** symbol. For this particular Data, you'll use a prompt to gather information from a user.

After this is a **Predefined Process** symbol. This symbol indicates that there is a function that needs to be performed. The support detail tells us the name of the function we need to look for. For simplicity, this first one is simply called Procedure in the flowchart. This symbol indicates that control of the code is passing to another process and won't return to our main code until that process is complete.

To see what the flow will be in this other process, we need to look for a flowchart with the same support details. That is, the separate flowchart to show the flow of the procedure should begin with a Start oval that contains the word Procedure. And, if you look at the Project 2 flowchart, you'll see that we do, indeed have that!

The Procedure Flowchart



The above flowchart indicates the logic for the procedure. You also can see that there is a Data symbol containing the support detail, **Prompt Argument**. This indicates two things. The first is that we have to send an argument to the procedure, and the second that you should be passing the data input by the user via the prompt (it can be either string or number, but it should be parsed first if it is a number). You do this in the code when you call the function. So, let's assume you're going to use the name myProcedure as the name of your procedure and you want to pass the data from a prompt that is stored in a variable called myPrompt. You would write the code like this:

```
myProcedure(myPrompt);
```

The value of myPrompt will be passed to the procedure as the argument.

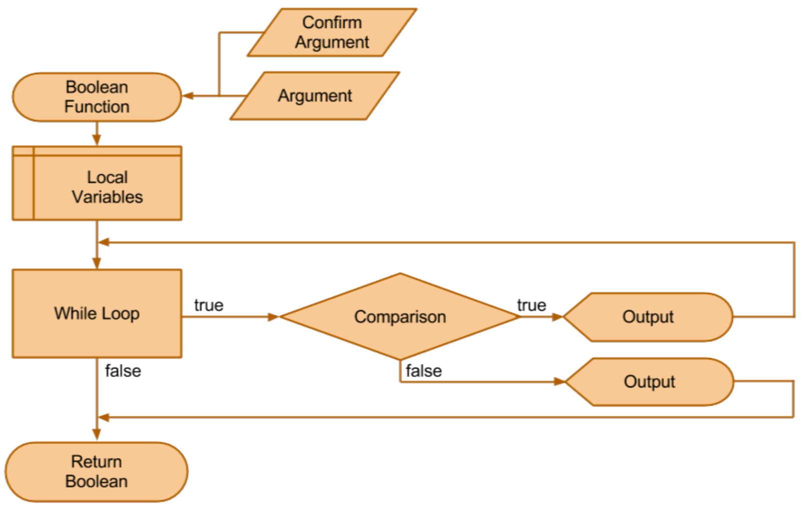
Notice, also, that the first symbol within the procedure flowchart is a **Conditional** symbol and that the support detail reads **Argument Conditional**. This means that the first thing that should happen in your procedure code is an `if` statement that somehow uses the argument that was passed in. So, if we use the previous example of passing an argument that is the value of myPrompt, your first step in the procedure will be to compare that value to something, such as a value. Then, you need to send some kind of output to the console.

The final symbol in the Procedure flowchart is the **Back** symbol. This indicates that the procedure is finished and control is passed back to the main code. This indicates that we need to go back to the main flowchart to see what comes next.

If you look at the flowchart for our main code, you'll see that the next symbol in the flowchart is another Data symbol indicating that we should have a confirm. This will allow you to gather a Boolean value from the user. After that is another Predefined

Process symbol with the support details of Boolean Function. Remember that this indicates that control of the code will flow to a separate function. Do we have a flowchart that begins with an Terminator symbol with support details indicating it's a Boolean Function? Yes, we do!

The Boolean Function



This function may look overwhelming, but that's only because of the way the arrows are represented. The logical flow is just like all the rest. If you follow the arrows, you'll see where the flow goes. And, like the other charts, this one begins with the Terminator symbol but has two Argument symbols pointing into the symbol. This means that you will need to send two arguments to this function. One will be the confirm data you gather from the confirm, and the other can be whatever data you'd like to use (string, number, or Boolean). This also requires you to declare the function so that it can accept two arguments.

Instead of having just the `if` conditional, this function enclosed that conditional inside of a while loop. This means that you'll perform a loop, and in the process of running the code in the loop, you'll check a conditional. If that condition is true, the loop runs again. But, if the condition is false, it will jump out of the loop using a `break;`. This makes it a bit more tricky to code.

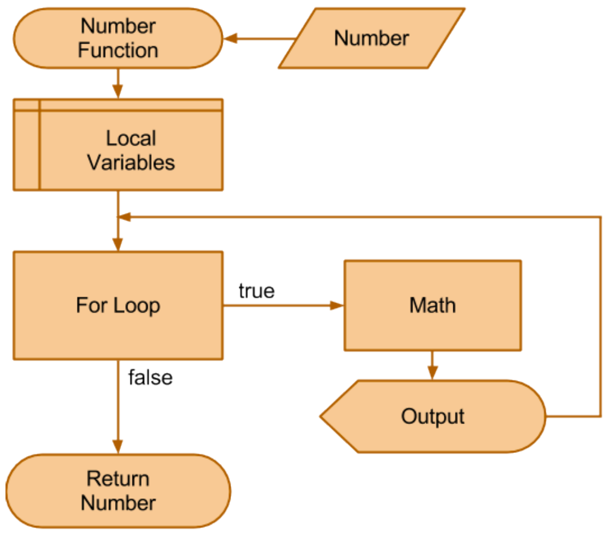
The next symbol is the Terminator for the function but rather than simply send control back to the main code, we also want to **return** a value from the function. This means we are sending a value out of the function and back to the main code for use. You can do this by calling the function from a variable assignment. So, if you have declared a variable **myReturn**, you can call the function like the following:

```
myReturn = booFunction(myConfirm, 5);
```

The value returned from the function will be stored in the variable **myReturn**. Also, notice that the Terminator symbol reads, **Return Boolean**. This means that your Boolean function must return a Boolean value; so in our example, myReturn will be either true or false.

Once control has been returned to the main code, the next symbol is another Predefined Process symbol, which indicates that the **Number Function** is the next to be called.

The Number Function



The first thing you should notice about this function is that the Argument symbol pointing into the Number Function terminator says **Number** instead of Argument. This is because you are required to send a number as an argument to this function. Similarly, you'll notice that the Return must also be a number.

This function contains a `for` Loop, which will require you to perform some kind of math function during the course of the loop followed by an output. In other words, every time the loop runs, some kind of math will be perform and an output will be sent to the console.

Once the Return is performed, control returns back to the main code and the next function is executed.

One very important thing to remember is that a function can only return a single value. That is, it can return a single string, a single Boolean, or a single number. And once the code hits the `return`, the flow of the code is returned to the main code. If you have any code after the `return`, it will not run because control no longer belongs to the function; it has been returned to the main code.

The flow of the Project 2 flowchart continues to another Predefined Processes and the function operates similar to the ones already detailed. To leave some of the fun up to you, see if you can figure out how the rest of your code will need to be written in order to follow the additional flow.