

# Decisions, Decisions: Learning Logic with Popfly

Which of your Facebook friends are from Washington? How can you create a mashup to find laptops from different manufacturers that cost under \$800? What are the top HipHop songs on iTunes? How can a mapping mashup use different colored pushpins to plot the cost of a gallon of gasoline across the United States?

Each of these tasks requires using logic to select just those items that meet the criteria, or to display different outcomes based on their values. This lesson will describe how to create Popfly mashups to create solutions for these kinds of problems.



Prepared by Mark Frydenberg  
Computer Information Systems Department  
Bentley College, Waltham, MA  
mfrydenberg@bentley.edu

# Decisions, Decisions: Learning Logic with Popfly



## Professor Popfly Mashups Referenced in this Lesson:

- MonsterMash (<http://www.popfly.com/users/professorpopfly/MonsterMash> )
- NorthSouth States (<http://www.popfly.com/users/professorpopfly/NorthSouthStates> )
- FacebookFriendsFilter (<http://www.popfly.com/users/professorpopfly/FacebookFriendsFilter> )
- Shopping (<http://www.popfly.com/users/professorpopfly/Shopping> )
- ConditionBuilderNumericComparisonTest (<http://www.popfly.com/users/professorpopfly/ConditionBuilderNumericComparisonTest> )
- PenniesFromHeaven (<http://www.popfly.com/users/professorpopfly/PenniesFromHeaven> )



## Learning Outcomes

After completing this lesson, you should be able to:

- Understand the different operators used for comparing values
- Explain the difference between branching logic and filtering logic
- Use the ConditionBuilder block to create conditions for the Filter and Logic blocks
- Use the Filter block to select a subset of data that match a condition
- Use the Logic block to select an outcome based on a condition or set of conditions
- Use AND, OR, and NOT operators on conditions
- Understand the difference in results when comparing numeric characters as characters vs. numeric values

## Overview

Which of your Facebook friends are from Washington? How can you create a mashup to find laptops from different manufacturers that cost under \$800? What are the top HipHop songs on iTunes? How can a mapping mashup use different colored pushpins to plot the cost of a gallon of gasoline across the United States? Each of these tasks requires using logic to select just those items that meet the criteria, or to display different outcomes based on their values. This lesson will describe how to create Popfly mashups to create solutions for these kinds of problems.

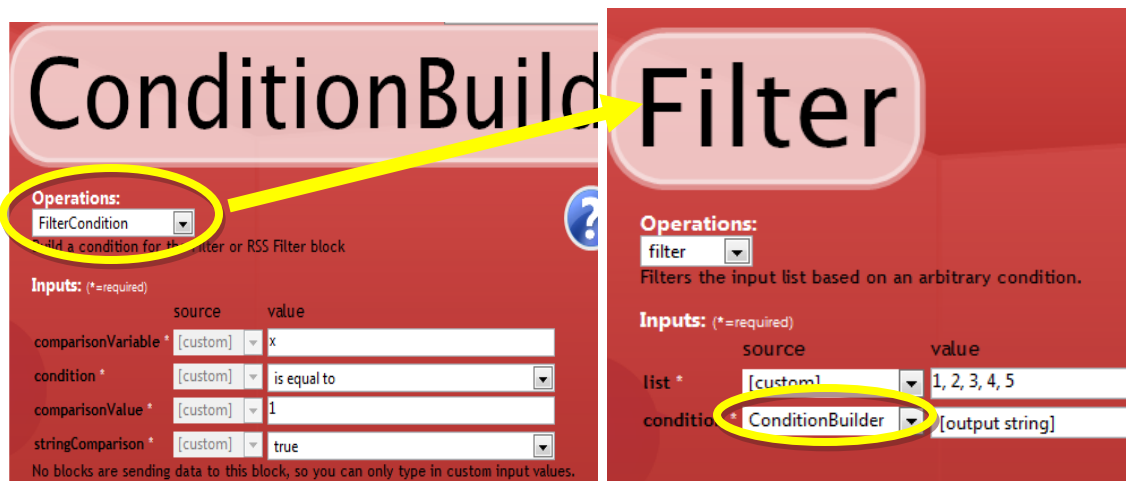
The lesson focuses on the use of three blocks: ConditionBuilder, Filter, and Logic. The ConditionBuilder block facilitates creating conditions (criteria for comparing values, such as “is this value equal to that one”) that can be used as inputs with either the Filter or the Logic block.

Popfly uses the Filter block to filter a list of data, returning only those items that meet the filter condition, and the Logic block to perform “if then else” and “if then else if” style branching logic using one or more conditions.

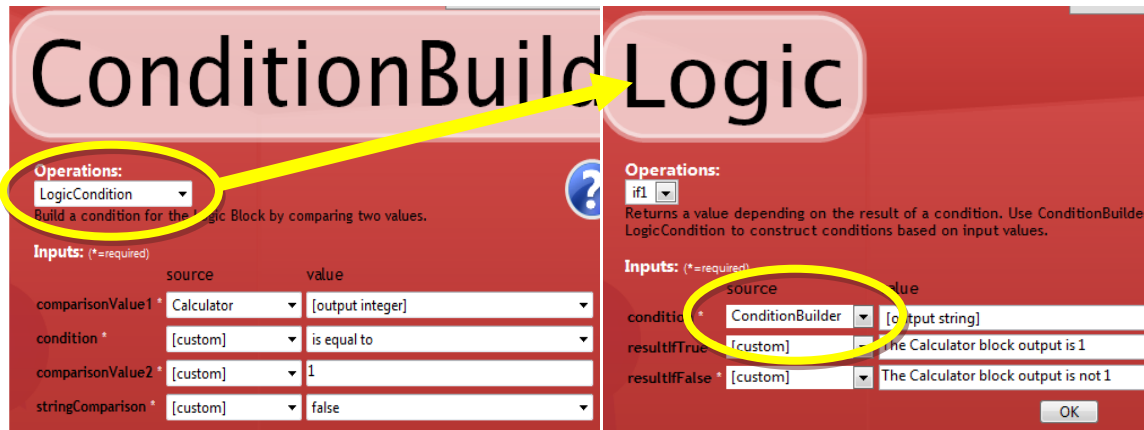
## ConditionBuilder Block

Use the Condition Builder block to create conditions to compare data values. The ConditionBuilder block has two operations for creating conditions, depending on how the condition will be used:

The **FilterCondition** operation creates a condition for use with the **Filter** block. Use it to create a condition to compare against every item in a list. An expression in terms of the variable x in the comparisonVariable input field represents each item in the list to be compared.



The **LogicCondition** operation creates a condition for use with the **Logic** block. Use it to compare two values and make a choice based on the result of the condition.



ConditionBuilder allows you to compare values using these conditions:

- is equal to
- is not equal to
- is greater than
- is greater than or equal to
- is less than
- is less than or equal to
- contains
- does not contain
- begins with
- ends with

Each of these operators may be applied to either character or numeric data. The default is to use a string comparison; that is, comparing character data. The last four operators (contains, does not contain, begins with, ends with) automatically treat their data as characters. The standard comparison operators (equal, not equal, greater than, greater than or equal, less than, less than or equal to) may be applied to numeric or character data.

The final input to a FilterCondition or LogicCondition is a Boolean value (either true or false) to tell Popfly how to interpret the data you are comparing. If it is character data, the value for the stringComparison input parameter should be true. If the data is numeric data, set that value to false. The default is to use a string comparison; that is, comparing character data.

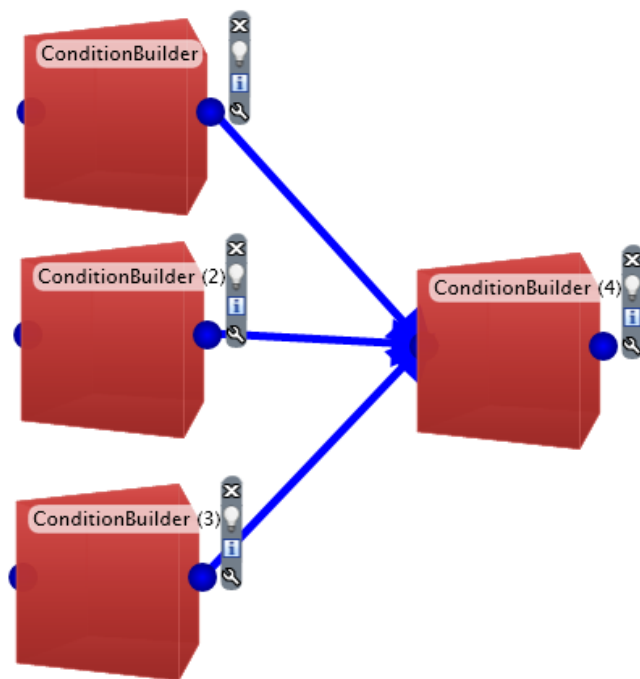
### Combining Conditions

The ConditionBuilder block includes operations Combine2Conditions and Combine3Conditions to build more complex conditions by combining two or three simple conditions created using the LogicCondition or FilterCondition operations.

You can combine simple conditions using AND or OR operators. In order for a set of conditions combined with AND to be true, each of the individual conditions must be true. For a set of conditions combined with OR to be true, any one of them may be true in order for the entire combined condition to be true. The Not operation negates a condition.

In this example, the 4-block sequence shown creates a FilterCondition to determine images that are GIF, JPG, or PNG files by checking to see if the associated url's end with .gif, .jpg, or .png.

To use this sequence in a mashup, one would connect ConditionBuilder(4) and a block containing the list of URLs to a Filter block.



## ConditionB

**Operations:**  
FilterCondition  
Build a condition for the Filter or RSS Filter block

**Inputs: (\*=required)**

	source	value
comparisonVariable *	[custom]	x
condition *	[custom]	ends with
comparisonValue *	[custom]	.gif
stringComparison *	[custom]	true

No blocks are sending data to this block, so you can only type in custom values.

OK

## ConditionB

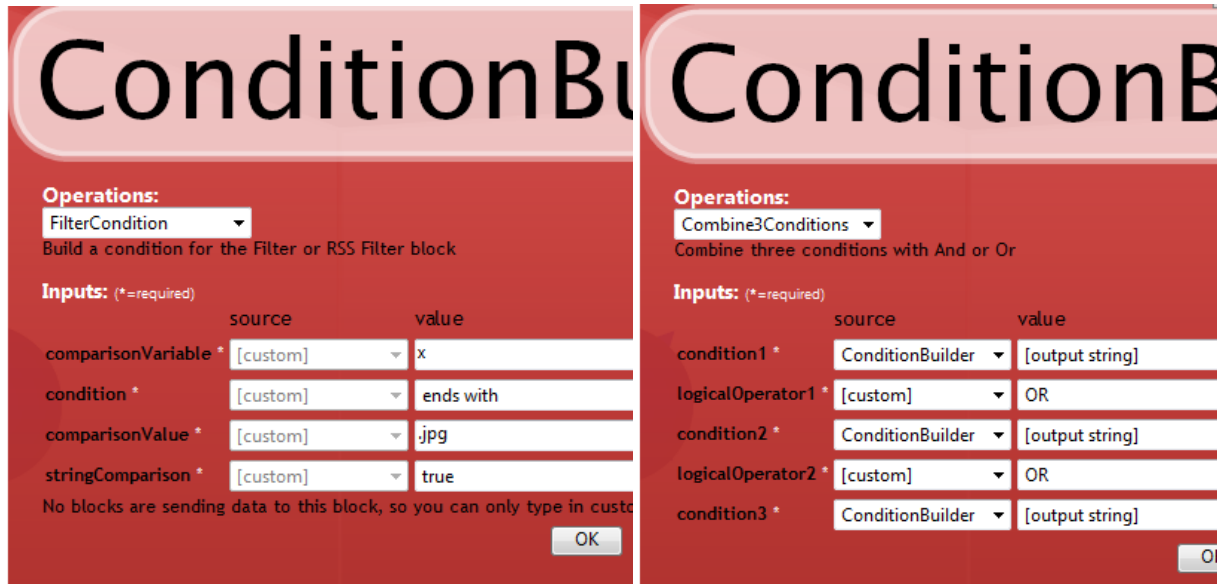
**Operations:**  
FilterCondition  
Build a condition for the Filter or RSS Filter block

**Inputs: (\*=required)**

	source	value
comparisonVariable *	[custom]	x
condition *	[custom]	ends with
comparisonValue *	[custom]	.png
stringComparison *	[custom]	true

No blocks are sending data to this block, so you can only type in custom values.

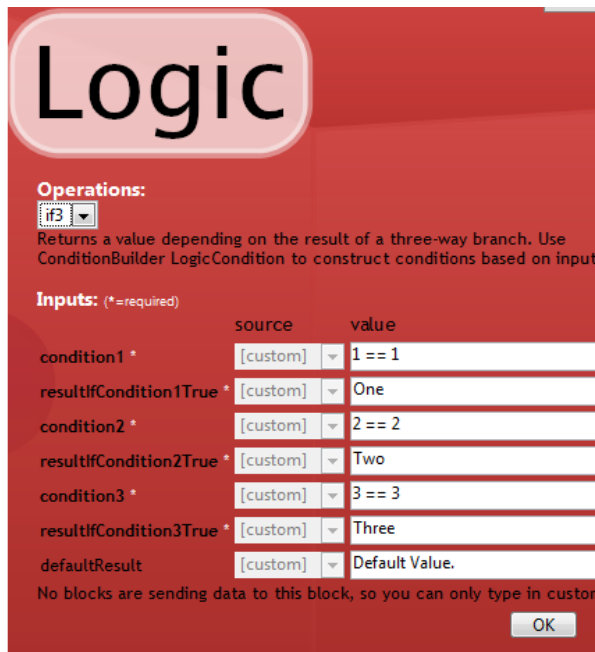
OK



## Logic Block

The Logic block determines a value based on branching following a series of comparisons. The simplest operation is if1 – an “if” operator using a single condition. If the condition is true, the operation returns the first value. Otherwise it returns the second value. Use the **LogicCondition** operation in the ConditionBuilder block when creating conditions to use with the Logic block.

The remaining operations, if2, if3, if4, if5, and if6 branch on 2, 3, 4, 5, and 6 conditions, respectively. The processing steps for the if3 operation of the Logic block are shown below. The other operations (if1, if2, if4, if5, if6) work similarly.



If (condition1) is true  
 Output the value resultIfCondition1True  
 Otherwise, if (condition 2) is true  
 Output the value resultIfCondition2True  
 Otherwise, if (condition 3) is true  
 Output the value resultIfCondition3True  
 Otherwise  
 Output the value defaultResult

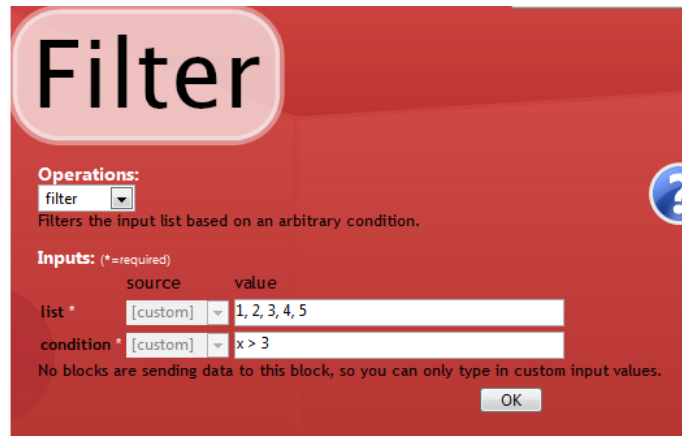
## Filter Block

The Filter block applies a filtering condition to a list of values (often specified by passing in an entire object from a connecting block) and returns those values of the list for which the filter condition applies. Use the **FilterCondition** operation in the ConditionBuilder block when creating conditions to use with the Filter block.

The processing steps for the Filter block are as follows:

For each item in the list to be filtered:

- Call the current item *x*.
- If *x* matches the criteria specified in the filter condition, include that item in results of the filter.



The condition for a Filter block must always be phrased in terms of a variable *x*, which stands for each item of the list to be compared. If you know JavaScript, you can write the condition in JavaScript. Otherwise use the ConditionBuilder block with a FilterCondition operation, to create the condition, and connect it to the Filter block.

The RSS Filter block works similarly, but the input is always an RSS feed, and that block allows you to specify the list within the RSS feed whose values you want to filter. For more information on how to use the ConditionBuilder blocks with RSSFilter, please see the RSS lesson in this series. If you know how to write JavaScript, you can write JavaScript code for the condition in the Filter block or RSSFilter block instead of using the ConditionBuilder block to specify the conditions.

## Logic Block Example 1: Monster Mash

This example describes how to create a mashup to display random images of monsters (or anything else!) in a tiled pattern. The “Tweak It” page shows the mashup and settings for each of the blocks used to create it.

Create Stuff ▾ My Stuff ▾ Help Stuff ▾ Overview Stuff ▾

Go! Save Reset More

Item 1: king kong  
Item 2: frankenstein  
Item 3: godzilla  
Go

**General Appearance:**  
 preferredBackgroundColor: #ffffff (Click to change.)  
 preferredTextColor: #000000 (Click to change.)

**Timer:**  
 frequency: 0.5  
 numberOfTicks: 20

**Calculator:**  
 upperBound: 3

**ConditionBuilder:**  
 condition: is equal to  
 comparisonValue2: 0  
 stringComparison: false

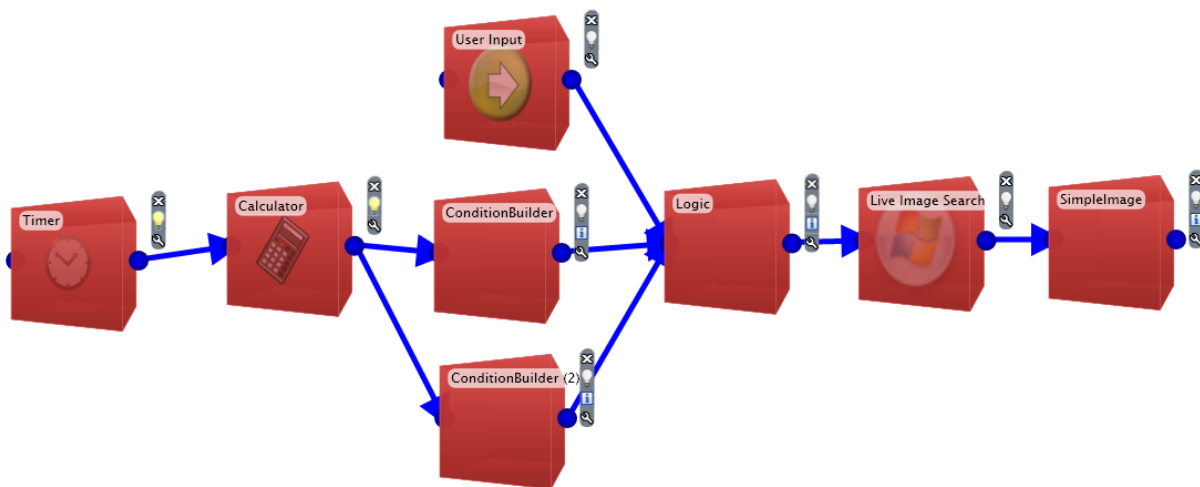
**ConditionBuilder (2):**  
 condition: is equal to  
 comparisonValue2: 1  
 stringComparison: false

**Live Image Search:**  
 count: 1

**SimpleImage:**  
 height: 100  
 width: 100

**User Input:**  
 label1: Item 1:  
 label2: Item 2:  
 label3: Item 3:  
 defaultText1: king kong  
 defaultText2: frankenstein  
 defaultText3: godzilla  
 buttonText: Go

Here's the mashup:

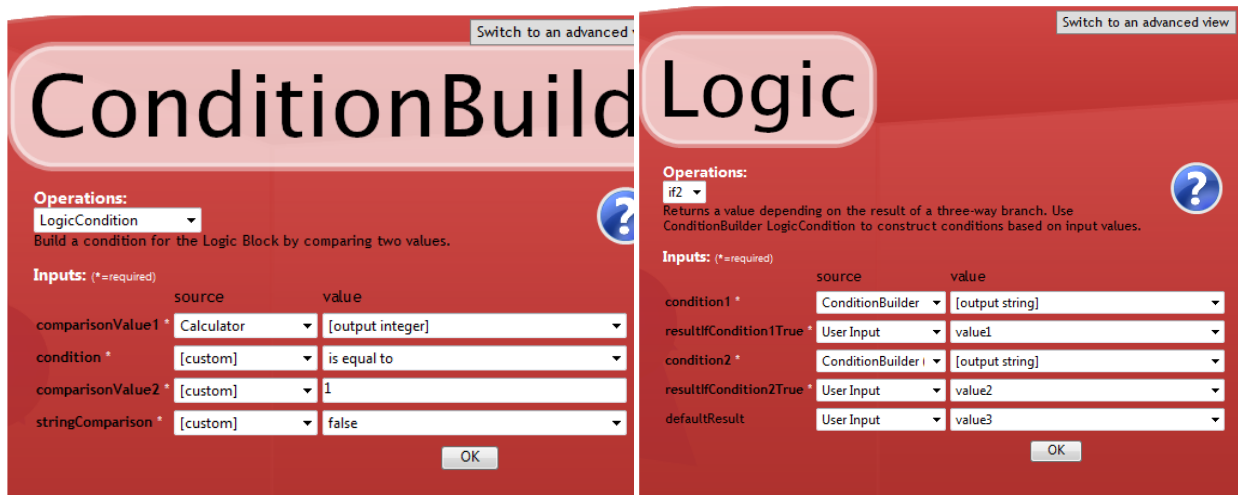


This mashup uses a Timer block invoke the block sequence that follows it every half-second for 20 seconds.

The Calculator uses the getRandomWholeNumber operation to select a random whole number between zero and three.

The two ConditionBuilder blocks use that value to create conditions using the LogicCondition operation to check its value. The only possible values are 0, 1, or 2. So two condition blocks are needed (to check if the value is equal to 0 or 1) The Logic block will use the default case if the value is 3.

The Logic block uses the if2 operation to check the value using the two conditions. Each value from the UserInput becomes a result of the condition, and describes an image to use corresponding to the value of the random number.



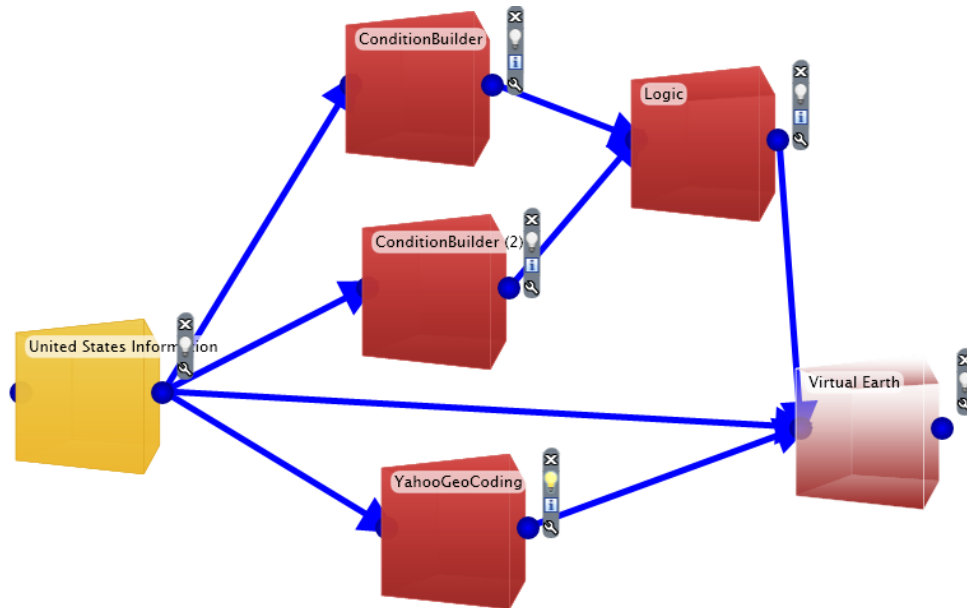
The Live Image Search block finds the first image whose description it receives from the Logic block. Finally, the SimpleImage block takes the URL of an image and displays it (with a specified height and width).

### Logic Block Example 2: North/South States

This mashup displays an up arrow icon for states that begin with North (North Carolina and North Dakota), a down arrow icon for states that begin with the word South (South Carolina and South Dakota), and an asterisk icon for the rest of the states.



Here's the mashup:



While the conditions here are arbitrary (“begins with North” or “begins with South”) a similar technique could be used to display red states and blue states, or colored pushpins based on other values. See the exercises, where you will use this technique to display a mashup of current gas prices on a map, coloring the pushpins based on the price of gas in each state.

## ConditionBuilder

**Operations:**  
LogicCondition  
Build a condition for the Logic Block by comparing two values.

**Inputs: (\*=required)**

	source	value
comparisonValue1 *	United States Information	State
condition *	[custom]	begins with
comparisonValue2 *	[custom]	North
stringComparison *	[custom]	true

OK

## Logic

**Operations:**  
if2  
Returns a value depending on the result of a three-way branch. Use ConditionBuilder LogicCondition to construct conditions based on input values.

**Inputs: (\*=required)**

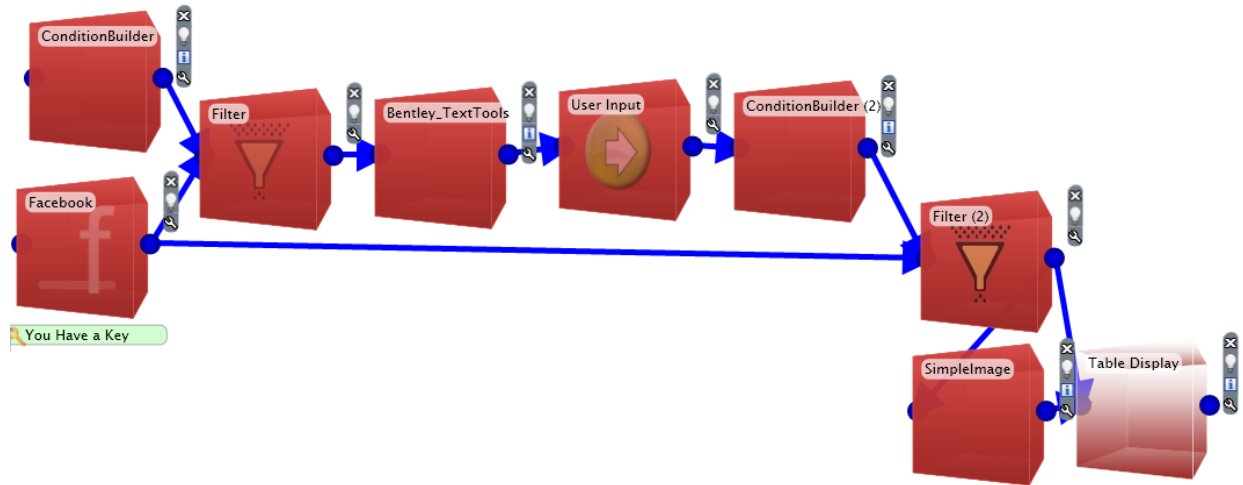
	source	value
condition1 *	ConditionBuilder	[output string]
resultIfCondition1True *	[custom]	http://icons.iconarchive.com/icons/icontexto/v
condition2 *	ConditionBuilder (2)	[output string]
resultIfCondition2True *	[custom]	http://icons.iconarchive.com/icons/icontexto/v
defaultResult	[custom]	http://icons.iconarchive.com/icons/icontexto/v

OK

## Filter Block Example: FacebookFriendsFilter

This mashup uses the Facebook block to display information about your Facebook friends from a given state. You will need to get a developer key when using the Facebook block if you want to show actual data about your Facebook friends. Click the link requesting that you get a key and follow the instructions to get one if you don't have a key already.

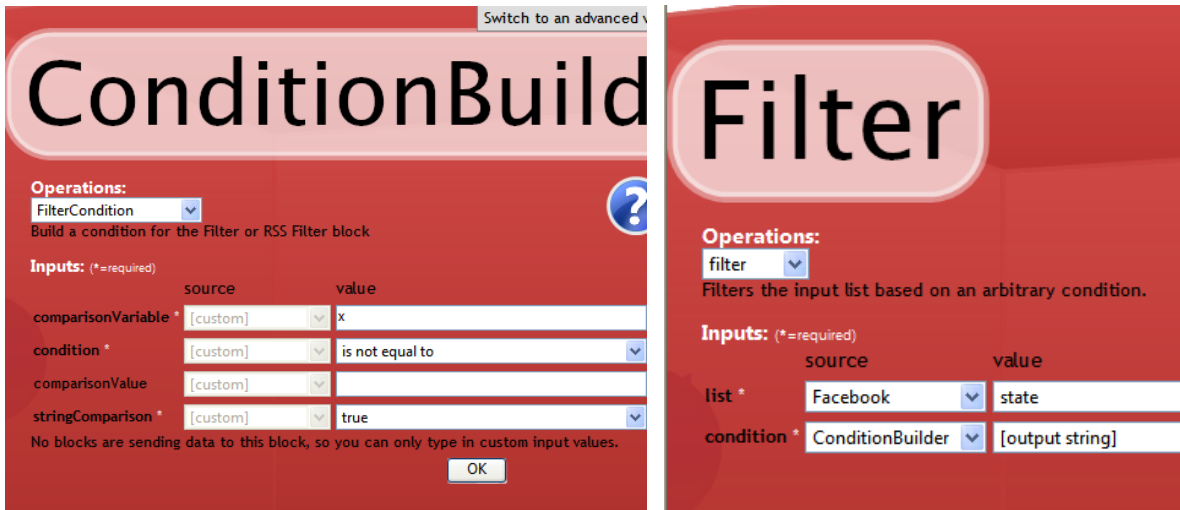
Each ConditionBuilder uses a FilterCondition connected to a Filter block.



The Facebook block's getFriends operation (with number of friends = 0) returns information about all of your Facebook friends.

This mashup uses two FilterConditions: first to return a list of all the states from all of the profiles which specify a state, and second, to find only those profiles which match the state that a user will select from a dropdown list in the User Input block.

The first ConditionBuilder checks to make sure that a state is provided by using the FilterCondition [x] [is not equal to] []. (The comparisonValue input is left blank in this case.) This is necessary because it is possible (and likely) that some friends did not specify a state in their Facebook profiles. For the purposes of this mashup, these friends will not be included, since they did not specify their state, and the mashup filters based on a person's state location. Here the value [x] is used as the comparison variable because the list being filtered in the Filter block is the list of state items from the Facebook block.



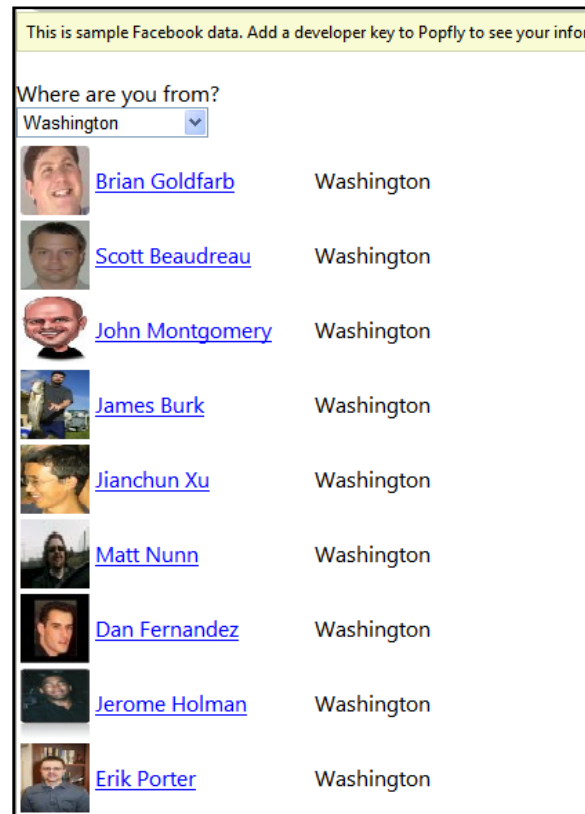
The Bentley\_TextTools block contains an operation called RemoveDuplicates that removes duplicate values from a list. This is needed so that each state appears in the dropdown list in the User Input block only once. (Without this block, if there were two friends from Massachusetts, the value of Massachusetts would appear twice in the dropdown list.) Removing the duplicate states creates a list of unique states by which to filter the list of Facebook friends.

As an aside, note that while this implementation doesn't do so, you might feed the output of the Filter block into the Sort block to sort the states alphabetically prior to displaying them as a dropdown list in the User Input block.

The second ConditionBuilder uses the filter condition `[x.state][is equal to][output string]` to filter the entire Facebook object based on the value of the state selected from the dropdown list in the User Input block. Note the use of `x.state` in the ConditionBuilder to specify a single entry in the entire Facebook object by which to filter the values.



The SimpleImage block creates the underlying HTML code to display the friend's thumbnail URL as an image, and the TableDisplay block displays the image, name, and state of each friend in a table, which will refresh every time you select a different state from the dropdown list.

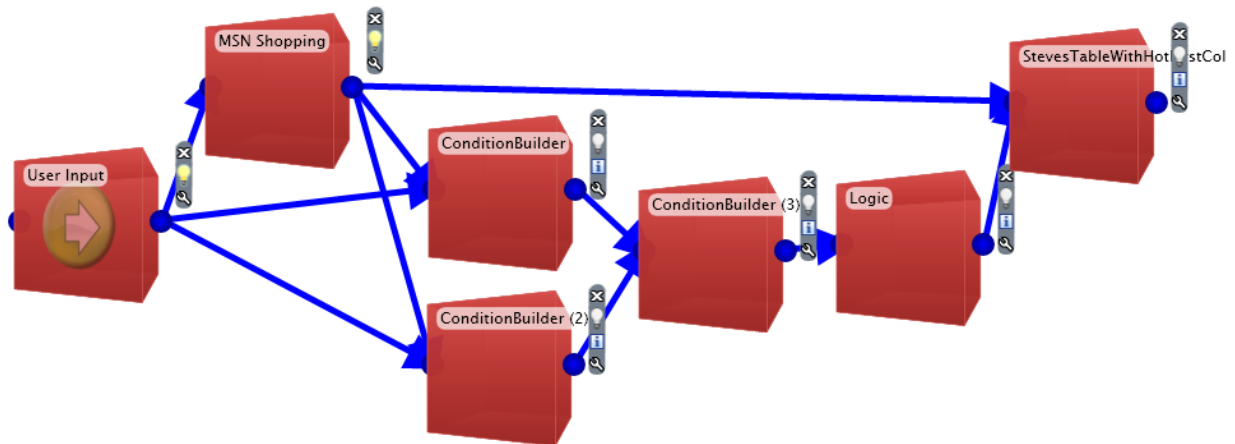


## Combining Conditions: Buying a Laptop

If you'd like to delve deeper into using logic operators, consider this example, showing how to use the AND logic operator to combine two simpler conditions.

Suppose you are looking for a Sony laptop that costs under \$800. You can use the MSN Shopping block to help you make your decisions.

The mashup uses the MSN Shopping block to search for items that match these criteria, and then displays a table showing the product name, price, a hyperlink to the product's URL, and word YES if the item meets both the brand and cost conditions, and NO otherwise.

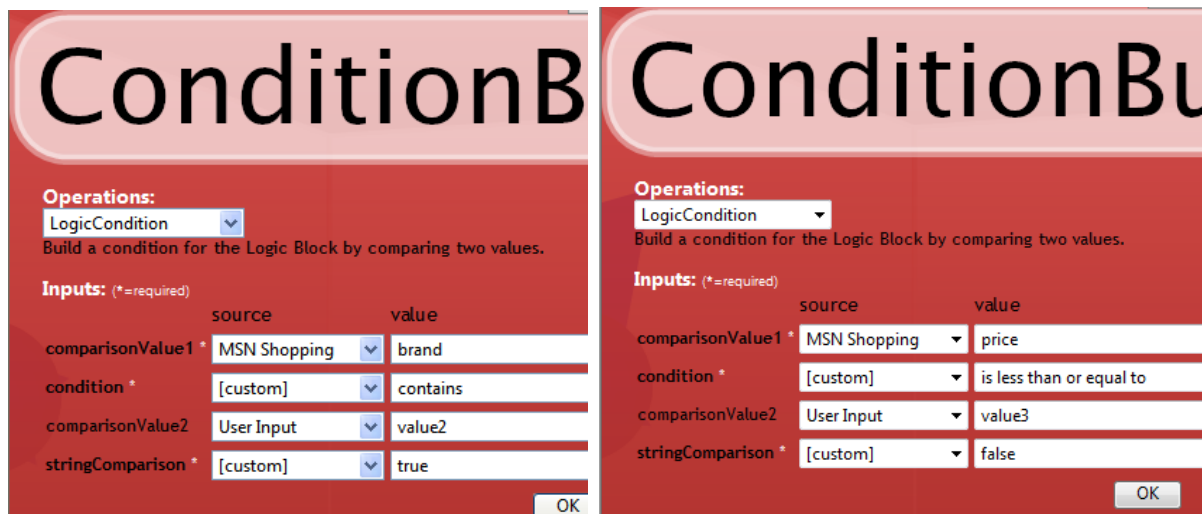


The user enters the brand, product, and maximum price to spend in the User Input block.

The getProducts operation of the MSN Shopping block is set to find information about 30 of the product specified that matches the brand and costs less than the specified maximum price. The default values on the User Input block search for laptops from Sony that cost less than or equal to \$800.

The first ConditionBuilder constructs the condition checking that the brand is Sony.

The second ConditionBuilder block checks to see if the price is less than or equal to \$800. Note that because this is a numeric comparison (the price is a number), the value of the stringComparison input to the LogicCondition operation is false.



The third ConditionBuilder block uses the Combine2Conditions operation to combine the two ConditionBuilder LogicConditions using an AND operator. The result of the combined condition is true if the results of both simple conditions are true. The Logic block checks the result of the combined condition, returning YES if it is true, NO otherwise.

# ConditionBuilder

## Operations:

Combine2Conditions ▾

Combine two conditions with And or Or

## Inputs: (\*=required)

	source	value
condition1 *	ConditionBuilder ▾	[output string]
logicalOperator1 *	[custom] ▾	AND
condition2 *	ConditionBuilder ▾	[output string]

OK

# Logic

## Operations:

if1 ▾

Returns a value depending on the result of a condition. Use LogicCondition to construct conditions based on input values.

## Inputs: (\*=required)

	source	value
condition *	ConditionBuilder ▾	[output string]
resultIfTrue *	[custom] ▾	YES
resultIfFalse *	[custom] ▾	NO

StevesTableWithHotFirstColumn displays the product name, price, URL, and whether or not it meets the criteria of being a Sony that costs less than or equal to the specified price.

Item	<input type="text" value="laptop"/>
Brand	<input type="text" value="Sony"/>
Costs <=	<input type="text" value="800"/>
<input type="button" value="Go"/>	

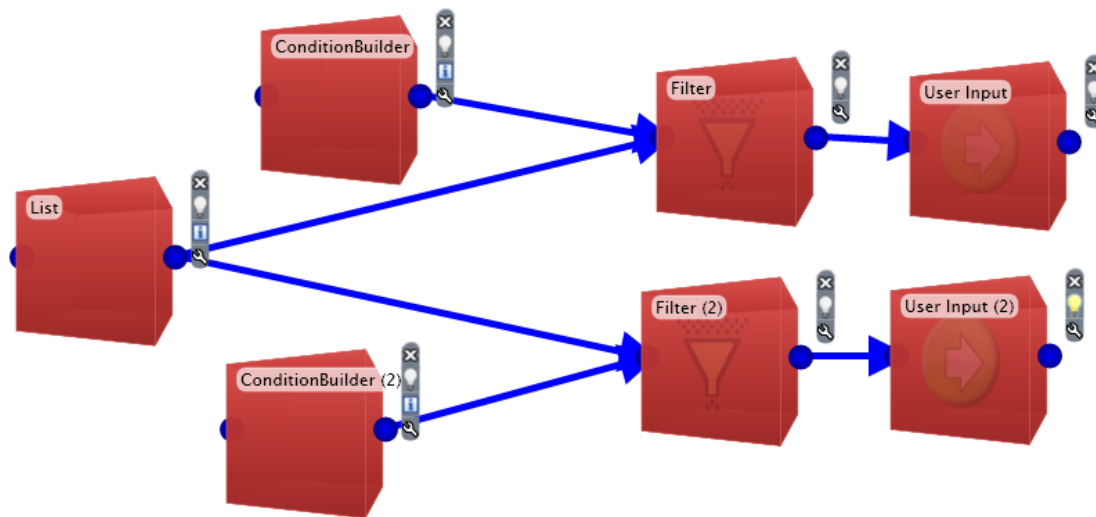
Item	Price	Meets Criteria?
<a href="#">HP Compaq Business Notebook nc4000 - Pentium M 1.4 GHz - 12.1" TFT</a>	395	NO
<a href="#">HP Compaq Business Notebook nc4010 - Pentium M 1.6 GHz - 12.1" TFT</a>	550	NO
<a href="#">Lenovo ThinkPad T61p 6460 - Core 2 Duo T7500 2.2 GHz - 15.4" TFT</a>	1614.89	NO
<a href="#">Sony VAIO NR160E/S - Core 2 Duo T5250 1.5 GHz - 15.4" TFT</a>	654	YES
<a href="#">Sony VAIO CR320E/R - Core 2 Duo T7250 2 GHz - 14.1" TFT</a>	1109.9	NO
<a href="#">ThinkPad 600 - PII 300 MHz - 13.3" TFT</a>	70	NO
<a href="#">Sony VAIO AR Digital Studio AR660U - Core 2 Duo T7500 2.2 GHz - 17" TFT</a>	1630.15	NO
<a href="#">HP Pavilion dv6345us Entertainment - Core 2 Duo T5300 1.73 GHz - 15.4" TFT</a>	929.99	NO
<a href="#">Apple PowerBook G4 - PPC G4 867 MHz - 12.1" TFT</a>	528.17	NO

## Technical Details: Comparing Character vs. Numeric Data

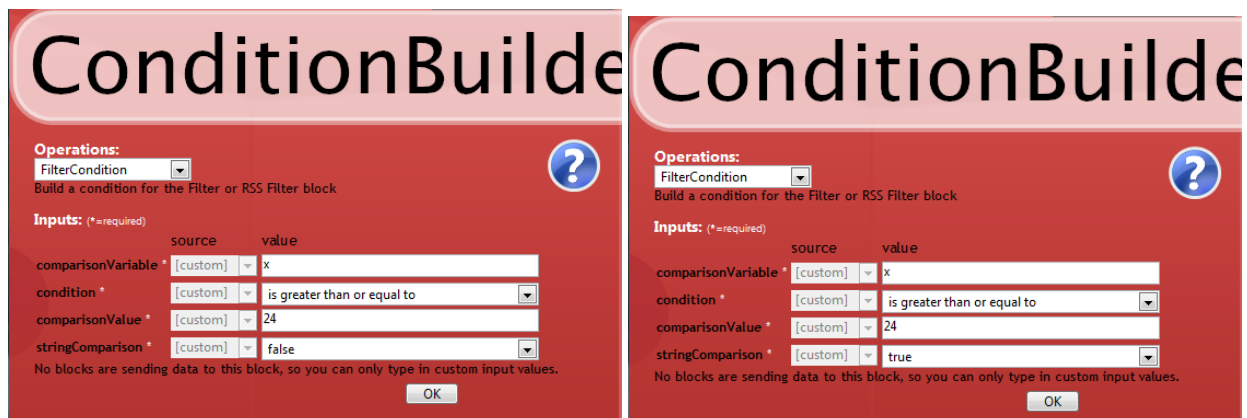
Read this section if you're interested in learning more about the internals of how JavaScript handles processing character and numeric data. Popfly blocks are written in JavaScript. Sometimes numeric data (such as zip codes) should really be treated as character data even though the values are strings of numeric characters.

For an example of this, view ProfessorPopfly's mashup "ConditionBuilderNumericComparisonTest". (<http://www.popfly.com/users/professorpopfly/ConditionBuilderNumericComparisonTest>)

This mashup that shows the different results for both string and numeric comparisons:

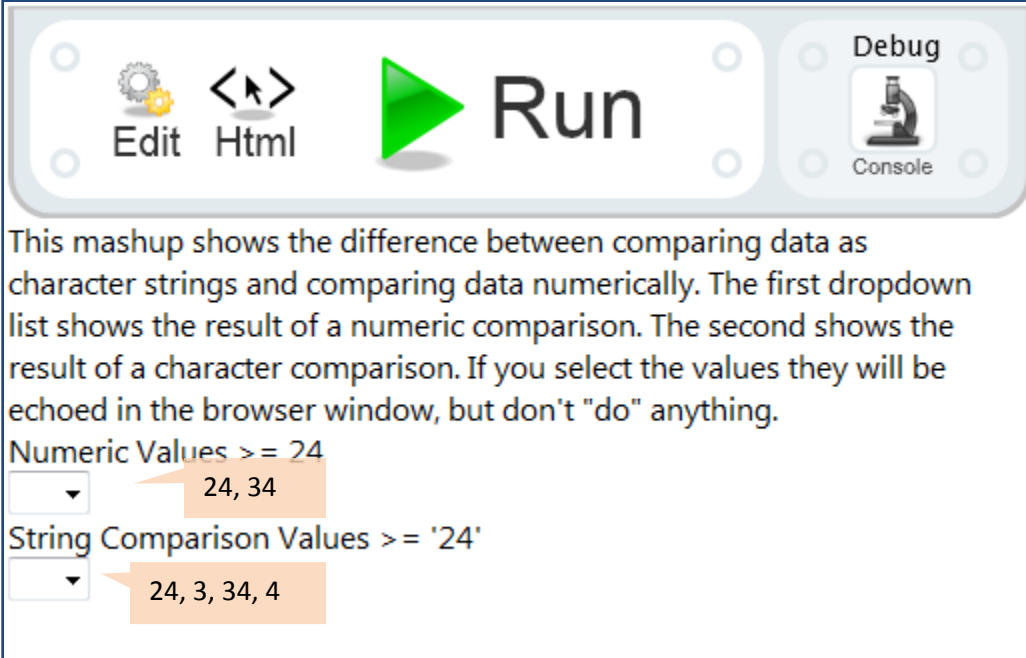


List creates a list with the values: 1, 14, 2, 24, 3, 34, 4. The mashup finds the values greater than or equal to 24. The FilterCondition operation is used because the condition is connected to a Filter block.



The only difference between the conditions in the ConditionBuilder blocks above is that the one on the left uses a numeric comparison while the one on the right uses a string comparison. Each block sends the comparison condition to Filter, which filters the list of values above using the designated condition.

When filtering the values numerically, the result of the condition ( $x \geq 24$ ) is 24 and 34. The results are displayed in dropdown lists within user input boxes.



This mashup shows the difference between comparing data as character strings and comparing data numerically. The first dropdown list shows the result of a numeric comparison. The second shows the result of a character comparison. If you select the values they will be echoed in the browser window, but don't "do" anything.

Numeric Values  $\geq 24$   
24, 34

String Comparison Values  $\geq '24'$   
24, 3, 34, 4

When filtering the values using a string comparison, the result of the condition ( $x \geq '24'$ ) is '24', '3', '34', and '4'. (Note the use of single quotes to indicate character string values.) A string comparison performs a character-by-character comparison of each list value against the character string '24'. The order for character strings to be compared is defined by a sequence known as ASCII which gives a distinct order to every alphanumeric character and symbol on a computer keyboard. Since 3 is further along than 2 in the ASCII sequence (the scheme by which computers represent the "order" of characters), 3 is included in the list, because the character '3' is further along in the list than the character '2' and anything that follows it (including '24') even though numerically,  $3 < 24$ .

## Popfly PopQuiz

1. Both the Filter block and the Logic Block make use of the ConditionBuilder block to set up conditions for filtering and branching, respectively. What is the difference between filtering and branching?
2. What does the expression in terms of a variable  $x$  represent when using a Filter Condition?
3. Suppose you are creating a mashup using a Popfly block that outputs a collection of image Urls. You want to display only those which are in GIF format. How would you use one or more ConditionBuilder blocks to test if the imageUrl represents a .gif file? Would you use a Logic block or a Filter block with the ConditionBuilder block(s)?
4. Suppose you are creating a mashup using a Popfly block that outputs a collection of image Urls. You want to use the BorderedImage block to display the GIF's with red borders, the JPG's with white borders, and the PNG's with blue borders. How would you use ConditionBuilder block(s) to set up these conditions? Would you use a Logic block or a Filter block with the ConditionBuilder block(s)?

## Try It in Popfly

Modify or build these Popfly mashups to demonstrate your understanding of this lesson. The more ducks, the bigger the challenge.



Using the UnitedStatesInformation block, several ConditionBuilder blocks, and the Filter Block, create mashups to display

- states whose bird is the hummingbird
- states that contain both e and o in their names
- states with two-word names

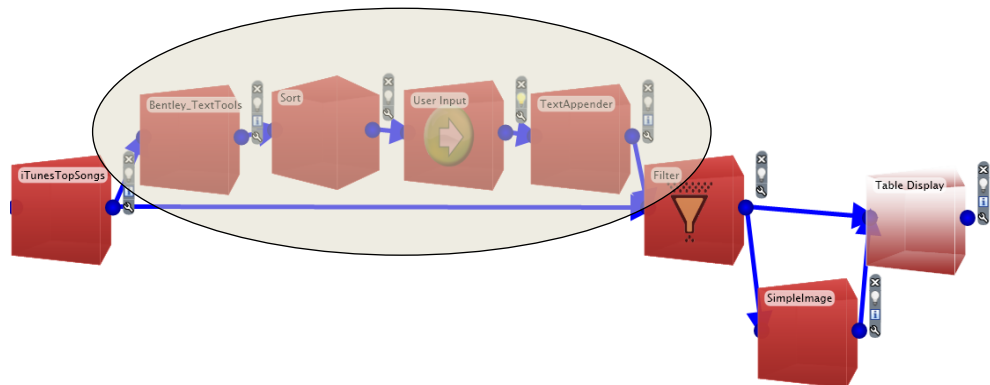


Build the mashup described in Popfly PopQuiz question 4 to display to use ImageScraper block or the Live Image Search block to obtain a collection of the images, and the BorderedImage block to display the GIF's with red borders, the JPG's with white borders, and the PNG's with blue borders.

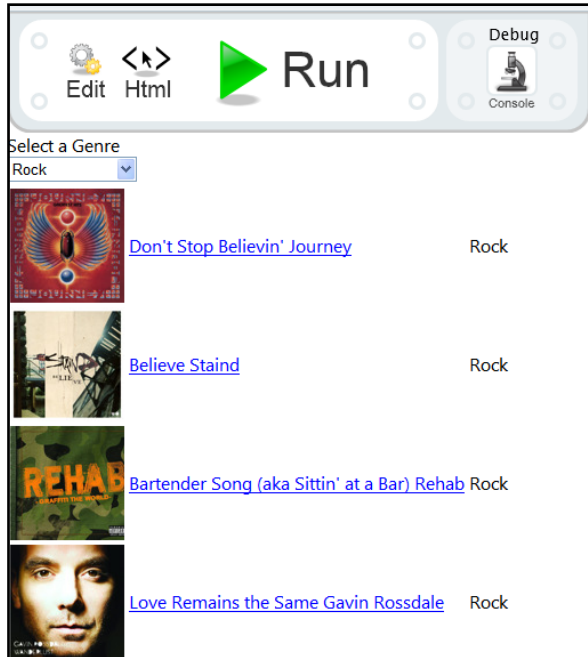


Find ProfessorPopfly's iTunesByGenre Mashup. This mashup uses the iTunesTopSongs block to display the most popular iTunes songs according to their genre.

The shaded block sequence shown below creates a dropdown list of unique music genres. The TextAppender block creates a JavaScript condition such as [ x.genre == 'Rock' ] where Rock is the value from the connected UserInput block.



Replace the TextAppender block with a ConditionBuilder block in which you create the same condition and connect it to the Filter block.



1. Write a mashup similar to the MonsterMash mashup, to simulate flipping a coin several times.

Use the Calculator block to select a random number that is either zero or one. If the value is zero, display the “heads” image below. Otherwise, display the “tails” image below.

Allow the user to specify the number of coins to flip by entering a value in a User Input block.

Use either the SimpleLight block or the SimpleImage block to display the image of the penny.



You can find these images here:

- [http://www.usmint.gov/kids/teachers/coinCurricula/images/01centCoin\\_obv.jpg](http://www.usmint.gov/kids/teachers/coinCurricula/images/01centCoin_obv.jpg)
- [http://www.usmint.gov/kids/teachers/coinCurricula/images/01centCoin\\_rev.jpg](http://www.usmint.gov/kids/teachers/coinCurricula/images/01centCoin_rev.jpg)

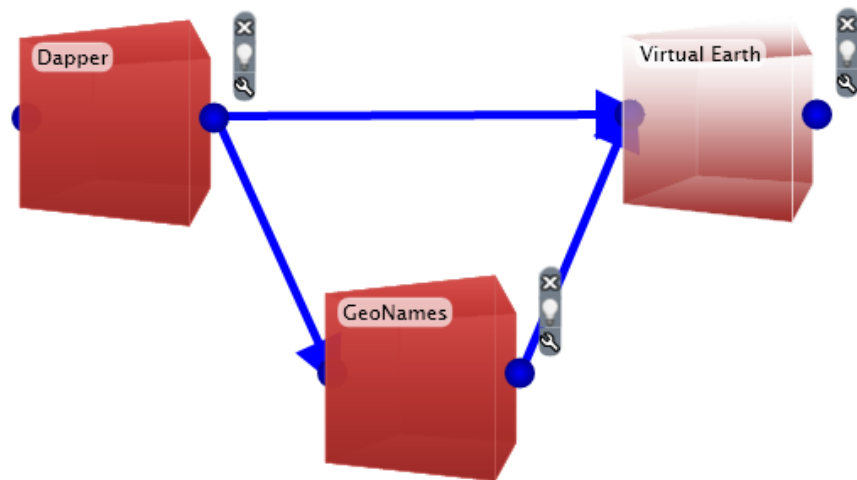
For added audio enhancement, use a Background music block to play Frank Sinatra singing the song “Pennies from Heaven.” The MP3 file is here:

- [http://www.ejazzlines.com/sample\\_mp3/LL-2109aa.mp3?EJAZZ=ithlc6u16gl1ka6snlcudtcub1](http://www.ejazzlines.com/sample_mp3/LL-2109aa.mp3?EJAZZ=ithlc6u16gl1ka6snlcudtcub1)

If you don’t know who Frank Sinatra is, you can skip this part. ☺ The solution in ProfessorPopfly’s PenniesFromHeaven mashup.

## 2. Open ProfessorPopfly’s GasPricesMap mashup.

This mashup uses a Dapper block to obtain data from the AAA website at <http://www.fuelgaugereport.com/sbsavg.asp> containing state-by-state average gas prices. You can find out more about the Dapper data mapper service at the web site <http://dapper.net>.



Remove the connection between Dapper and Virtual Earth; replace it with two ConditionBuilder blocks and a Logic block to display a different colored pushpin depending on the price of gas.

If the price of gas in that state is less than \$4.10, use a blue pushpin.  
If the price of gas in that state is less than \$4.250, use a red pushpin.  
Otherwise use a white pushpin.

Here are URL’s for three colored pushpin icons:

- <http://maps.google.com/mapfiles/kml/pushpin/red-pushpin.png>
- <http://maps.google.com/mapfiles/kml/pushpin/wht-pushpin.png>
- <http://maps.google.com/mapfiles/kml/pushpin/blue-pushpin.png>

Please note: At the time of writing this exercise, gas prices ranged between \$4 and \$5 per gallon across the country. If these values are out of date when you are completing this exercise, change them to be more appropriate!

## Learn More about Logic

- JavaScript comparisons ([http://www.w3schools.com/JS/js\\_comparisons.asp](http://www.w3schools.com/JS/js_comparisons.asp))
- ASCII Code and Character ordering (<http://en.wikipedia.org/wiki/ASCII>)