Relational Comparisons & Logical Operators in Alice

Even though Alice provides a variety of primitive Boolean functions, they still cannot cover all programming possibilities. The programmer must have the option to compose Boolean expressions according to the needs of their program.

To allow for more flexibility, Alice, along with other programming languages, provide the relational operators to compare the relationship between two values. These are very similar to mathematical concepts used to compare numeric values, and when programming, it is most common to compare numeric values this way.

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

Any expression involving a relational operator must be Boolean, meaning the expression must be either True or False. Quite often, we will use relational operators to compare the values of variables that we have created in our programs. For example, length > width.

This expression checks the value of length against the value of width. If length is greater than width, the expression is True. Otherwise, the expression is false.

Relational operators can be used to create conditions used in If/Else statements. They can be found in the world object's list of functions under the math category. Each operator has two operands, a and b. When you create a condition using a relational operator, you must specify values or variables or expressions for a and b.
Relational Comparisons & Logical Operators in Alice

Tutorial 4.2 – Using a Relational Operator

A big fish is in the water with two gumdrops, one of which is red and the other is yellow. These are magic gumdrops. If the big fish eats the red gumdrop, the big fish will become twice its size. If it eats the yellow gumdrop, it will shrink to half its size. The big fish swims to the gumdrop that is closest to it and eats it.

We will use the less than operator (<) to determine which gumdrop is closest to the big fish by comparing the distance to the red gumdrop and the distance to the yellow gumdrop. The pseudocode for the algorithm looks like this:

If (bigfish distance to red < bigfish distance to yellow)
    bigfish turns to face red gumdrop
    bigfish moves to red gumdrop
    bigfish eats red gumdrop
Else
    bigfish turns to face yellow gumdrop
    bigfish moves to yellow gumdrop
    bigfish eats yellow gumdrop
End If
bigfish turns to face the camera

1. A world for this tutorial has been partially created for you. Copy the GumdropBigfish world from the Handout folder. There is already a bigfish object, a redGumdrop object, and a yellowGumdrop object.

2. In the Method Editor, create an empty If/Else statement. For now, use True or False as a placeholder. We will replace this shortly.

We need the less than operator to compare the distance to the red gumdrop against the distance to the yellow gumdrop.

3. Select the world object and look under the functions tab for the \( a < b \) relational function. Drag this function to the condition of the If/Else statement. For now, we will need placeholders for \( a \) and \( b \), so use \( a=1 \) and \( b=2 \).

4. Replace the placeholder for \( a \) with the distance from the big fish to the red gumdrop. Select the
bigfish object and find the distance to function. Drag and drop this function over the 1 placeholder, and specify the redGumdrop as the target object.

5. Replace the placeholder for \( b \) with the distance from the big fish to the yellow gumdrop. Once again, select the bigfish distance to function, and drag it over the 2 to replace it, this time specifying the yellowGumdrop as the target object.

6. Add the instructions for each branch of the conditional statement. A basic version of the completed program is shown below. Consider each instruction carefully and ensure you understand what they accomplish.

```plaintext
// Determine which gumdrop is closest

// The red gumdrop is closest
bigfish ~ move to redGumdrop more...
bigfish ~ resize 2 more...

// The yellow gumdrop is closest
bigfish ~ move to yellowGumdrop more...
bigfish ~ resize 0.5 more...
```
Relational Comparisons & Logical Operators in Alice

You may have noticed that **comments** were added to this program. As programs increase in complexity, comments are one way to help explain the program in plain language. They are notes from the programmer and should be used whenever the purpose or result of a section of code is not obvious.

**Boolean Variables**

Now that our Boolean conditions are become longer and more complicated, it is worth revisiting the idea of the Boolean variable. Just as we can use a numeric variable to store a complex calculation, we can use a Boolean variable to store a complex condition. This becomes even more important in Alice, since long expressions tend to run off the screen.

Consider the example from Tutorial 4.2. We compared the distance of the big fish to the red gumdrop and yellow gumdrop. Although this only involved a simple comparison, the statement in Alice was quite long.

A Boolean variable could be used to store this result, and with a meaningful name, would result in much clearer code.

At first glance, this may actually seem like more work. In fact, this example is right on the edge of usefulness. Both ways to write the code have advantages, and both are equally good. As the code increases in complexity, however, the use of Boolean variables will, in the end, greatly improve your programming.