Chapter 1

EyesWeb Flow Control

EyesWeb provides blocks to manage the flow of data inside a patch. This chapter describes such blocks.

1.1 For Loop

The ForLoop is a type of subpatch which iterates the execution of the block in a patch for a number of times. Its main purpose is to use standard blocks to deal with items in a collection, which otherwise could not be done unless developing custom blocks. The use of the ForLoop block is explained by referring to patch for_loop.eywx which is distributed with EyesWeb and is depicted in Fig. 1.1 and 1.2.

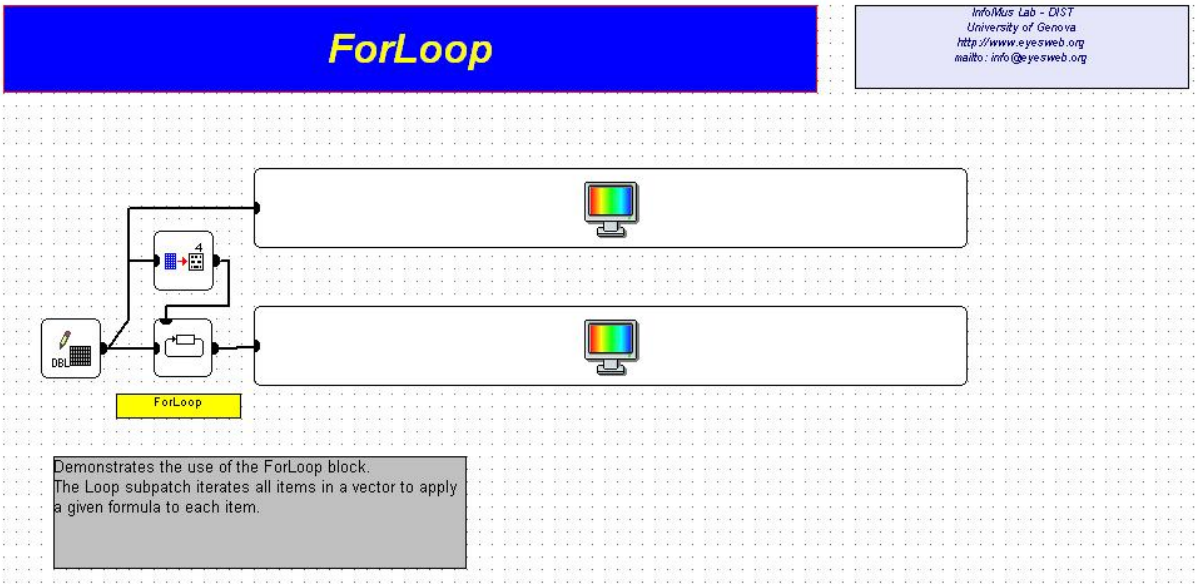

Figure 1.1: The tutorial patch to explain the ForLoop block.

The patch is quite simple: a vector is generated and a FunctionEvaluator block is applied to each item of the vector. Note that the FunctionEvaluator block does not deal with vectors, but it deals with scalar items; thus, the ForLoop in this case is the only way to apply the FunctionEvaluator block to the vector, as there is no such block which

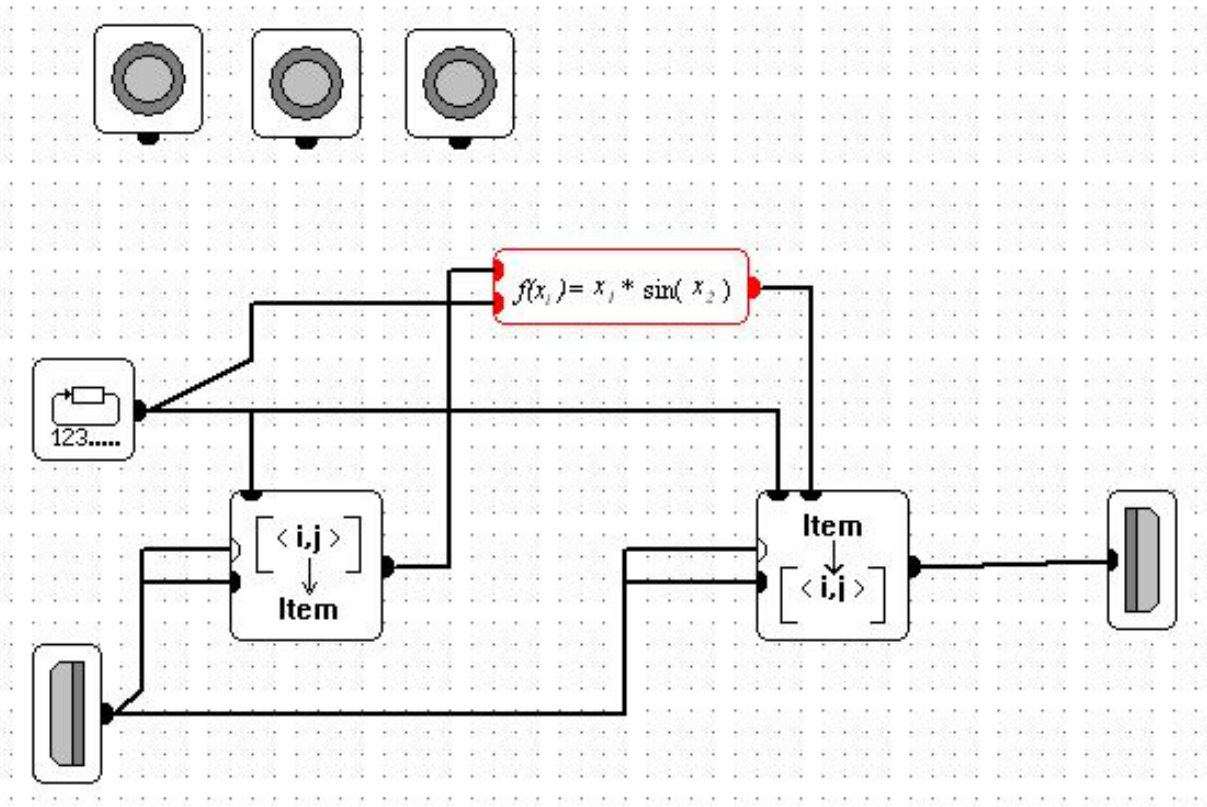

Figure 1.2: The ForLoop subpatch used in the ForLoop block tutorial.

natively works with the vector datatype. The following is a sketch of the pseudo-algorithm implemented by the patch:

- Generate a new vector of size N
- For each i item in the vector
  - Extract item i from the vector (note: the extracted item is a scalar number)
  - Apply FunctionEvaluator block to such scalar item
  - Insert the scalar result as item i of the vector

The algorithm is quite straightforward, let's not analyze how it is implemented as an EyesWeb patch. The first block on the left (see Fig. 1.3) generates a vector of floating-point numbers. The user can specify the values of each item in the vector by setting them in the parameters of the block. Following the generator block there are other two blocks: the MatrixGetDescription block (Fig. 1.4) and, below it, the ForLoop block (Fig. 1.5). The MatrixGetDescription block simply extract the size of the vector, i.e., the number of columns; the result is passend as a paramater to the ForLoop block, as the EndIndex. As a matter of fact, the ForLoop block can be controlled by specifing the start index of the loop, the end index, and the step by which the index is increased at each iteration. Note that the loop is executed with index going from Start Index to End Index - 1 (i.e., the Start Index is included, the End Index is excluded). The ForLoop block is a subpatch, thus, double-clicking on it will open another content which represents the blocks which are iterated. The subpatch is visible in Fig. 1.2.

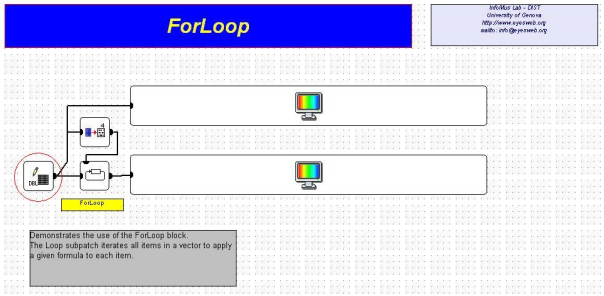

Figure 1.3: The Vector generator block.

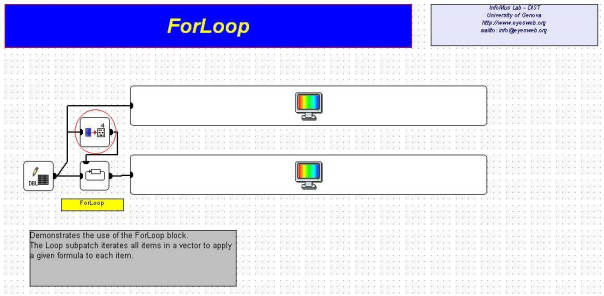

Figure 1.4: The Vector generator block.

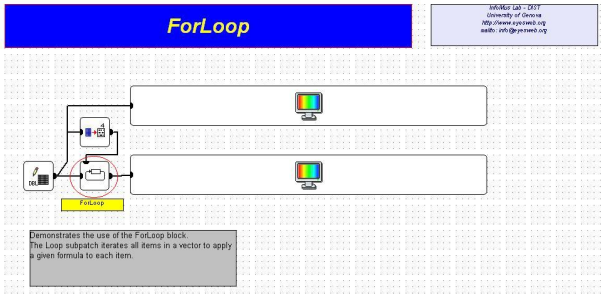

Figure 1.5: The Vector generator block.

The subpatch starts with a Subpatch Input Pin, i.e., the block on the left with a vertical rectangular icon (Fig. 1.6). This block provides the input data of the block inside the

subpatch, thus, the output of this block is the same as the input of the block, i.e., the input vector. However, at each iteration loop, the output can be different as it will be explained later. The output of this block goes to the GetItem block (Fig. 1.7), which extract an intem from the vector. The index of the item to be extracted is provided by the ForLoopIndex block (Fig. 1.8), the one above the input pin block, which, at each loop, provides the current value of the index as output, that is, 0 on first iteration, 1 on second iteration (assuming Step is set to one), etc. The index is passed as an argument to the GetItem block, which extract the given scalar value. The extracted value is given as input to the FunctionEvaluator block (Fig. 1.10), the FunctionEvaluator block also receives the loop index as input, as the function it computes is choosen by the user to be dependant on both the current value of the item in the vactor and the index. Of course, the user could have decided to use other inputs too (e.g., the current time, the value of item $i-1$ in the vector the size of the vector, etc.). Finally, the output of the FunctionEvaluator evaluator block is fed, as a parameter, to the SetItem block (Fig. 1.11) which receives the current vector as an input and changes the value of the given item. The output of the SetItem goes to the Subpatch Output Pin. From here, the resulting vector can be fed back to the input pin, which occurs on every iteration except the last one, or it can be sent to the output pin of the block, which occurs on last iteration.

Note that the feedback to the input pin is not the default behaviour, you have to specify it in the input pin. in facts, the input pin block has a parameter named Feedback pin, which is set to None by default. From the dropdown combo-box you have to select the pin from which to get the feedback[1][2].

It is noteworthy that, by means of the ForLoop block, you have the a sort of possibility to change, at runtime, the structure of the patch, which is usually not allowed in EyesWeb. In facts you may think of the content of the loop as a pattern which is repeated several times, and the number of times can change at runtime. Thus, the above loop iterated N times might be similar to the patch of Fig. 1.12[3]

As a final remark, let's look at the three unconntected parameter pins which are available inside the subpatch (Fig. 1.13). They have not been used here, but they might be useful in some cases. They represent the Start Index, the End Index, and the Step parameters. They might be used, for instance, to verify whether the current iteration is the first one (Start Index equal to ForLoopIndex) or the last one (End Index plus one equal to ForLoopIndex).

[1]Rationale for having the choice in the input pin instead of the output pin. Specifing this in the output implies a signature of the output pin with a variable number of parameters, which is sometimes a bit more tricky to be used.

[2]Rationale for not having an explicit graphical link from the output pin to the input. There are different reasons; firstly, the user is used to see input and output pins as starting or final blocks; thus, an outgoing link from the output pin or an ingoing link to the inpit ping might sound a bit strange. Secondly, the feedback is vary common in the loop and adding a graphical link to represent it might add confusion to complex subpatches; since the user knows that this normally happens, the price for the additional possible confusion does not worths.

[3]Just similar to that, not the same! If you use blocks that have an internal status, such as the counter block which keeps the current count internally, you have to keep in mind that the block instance is always the same. Thus, if you count the inputs, the counter is incremented at each iteration!

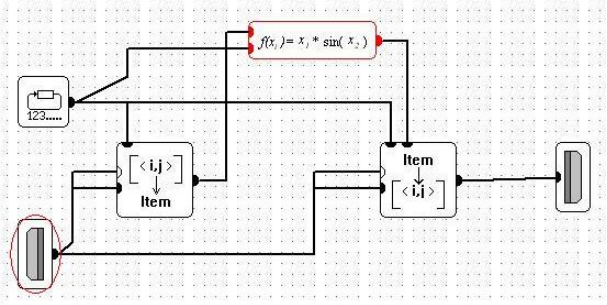

Figure 1.6: The subpatch input pin block.

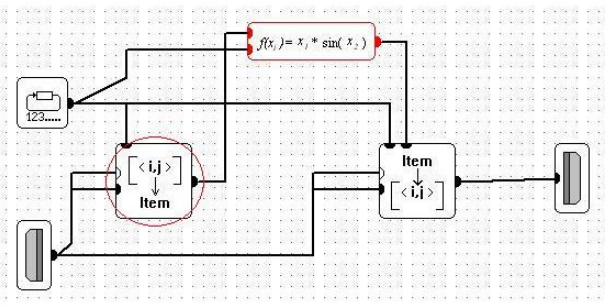

Figure 1.7: The GetItem block.

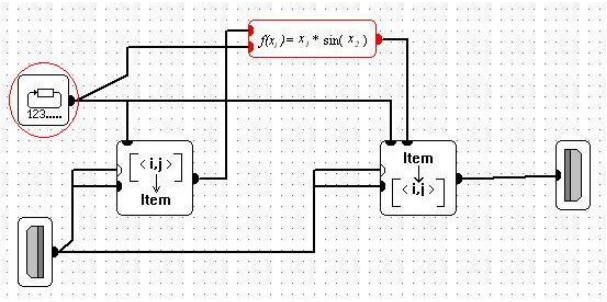

Figure 1.8: The iteration index block.

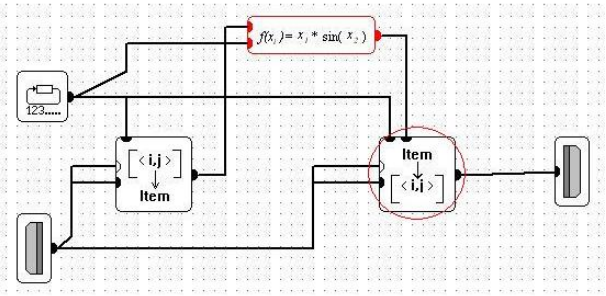

Figure 1.9: The SetItem block.

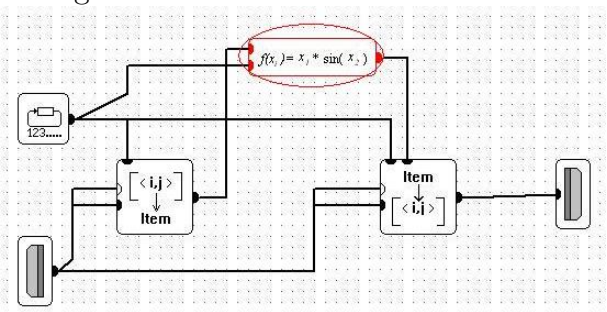

Figure 1.10: The Function Evaluator block.

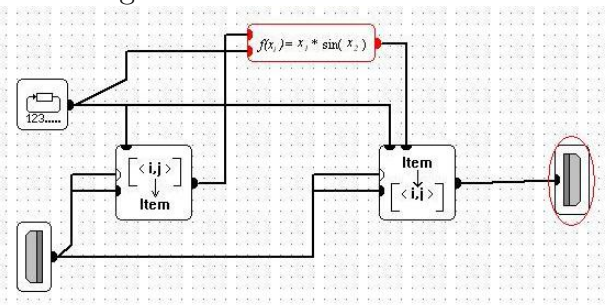

Figure 1.11: The subpatch output pin block.

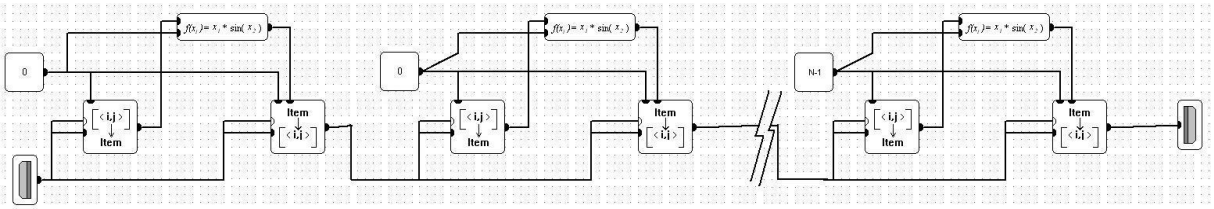

Figure 1.12: A sketch of the unrolling of the ForLoop subpatch.

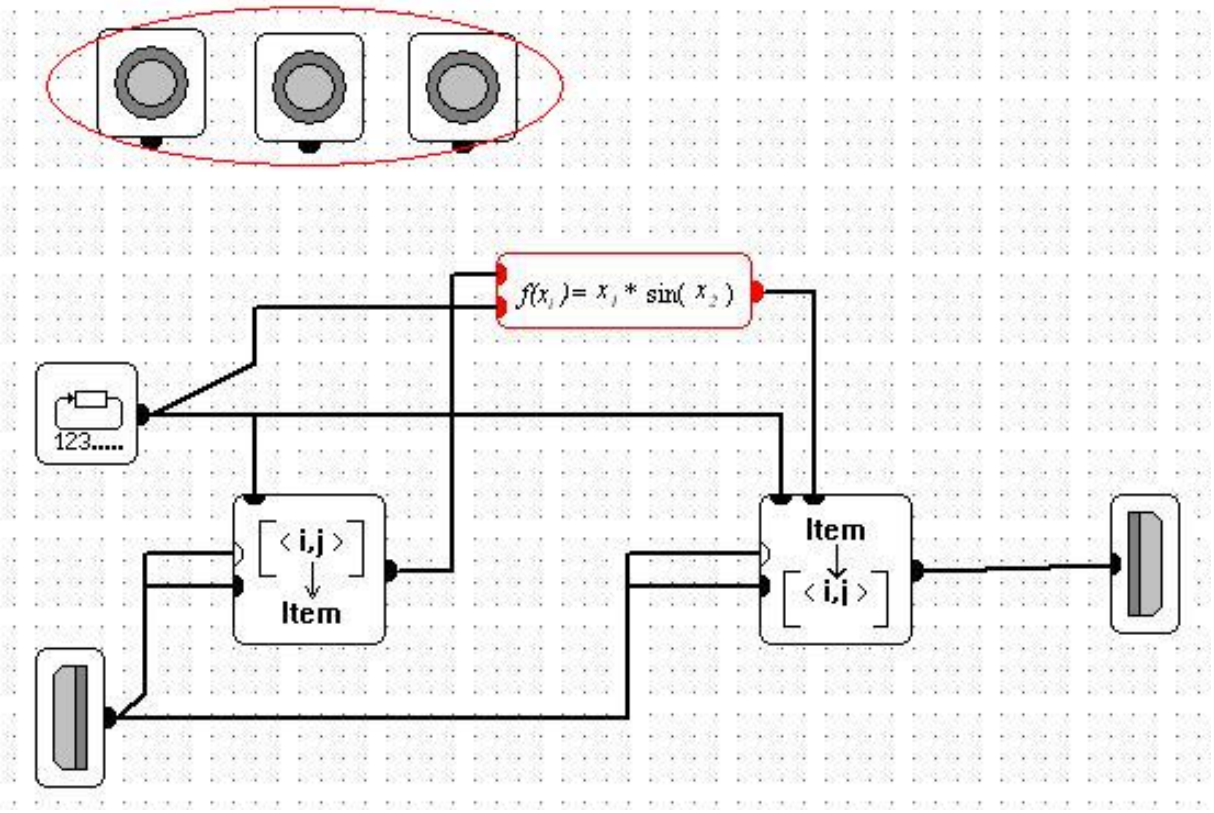

Figure 1.13: Addition parameters are available inside the subpatch to check whether the current iteration is the first or the last one.