NetsBlox Lesson: Introduction to Lists
Movie App

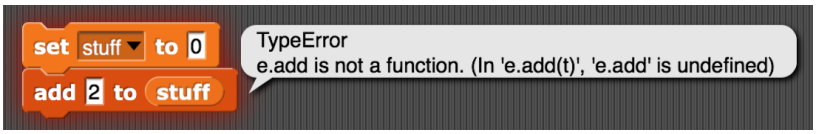
We have seen how a variable can store some value, like a number or some text. However, a variable can also contain a list of values (or items, as we typically say). That is a very powerful feature that is used pretty widely in programming. The variables tab has several blocks related to lists:

One important thing to remember is that we have to specify if we want a variable to contain a list. We cannot add an item to a variable if we did not make it a list first. This is what to do:

and the variable **`stuff`** will look like this after executing the block:

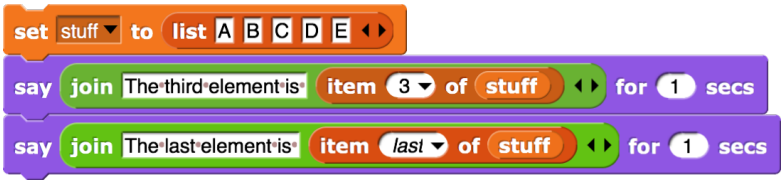
The arrow keys on the **`list`** block allows us to add or remove items. If we do not want to add items initially, we still need to make the variable a list. Simply eliminate all item placeholders from the **`list`** block by clicking the left arrow. This is what we should have:

If we forget to make the variable a list and try to add an element to it, we get a somewhat cryptic error message:



Notice the red highlight around the blocks to. If you see this, remember that the variable needs to be a list before we can add an item to it.

As you see above, to add an element to the end of a list, simply use the **`add`** block. To access individual elements, we have to use the index concept. You can simply think of lists as if their elements are numbered. At the beginning of the list is the first element or item #1, then comes the second or item #2 etc. Then we can simply access elements with the **`item`** block:



You should see C and then E after a second.

To see how long a list is, that is, how many elements it has, use the **`length`** block. That is useful when we want to iterate through all elements of a list as we'll see later. The other list blocks are fairly self-explanatory, so feel free to try them out on your own.

Our first exercise is to create a list of 10 random numbers:

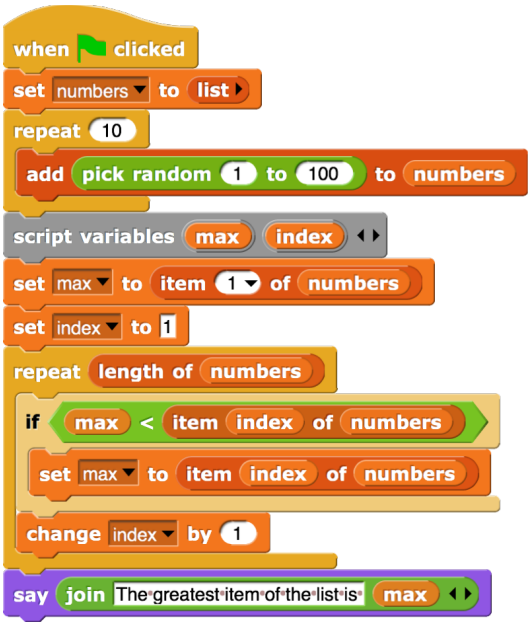
Not much explanation is needed, right? And this simple program illustrates the power of loops and lists: by simply changing the 10 to a million, we can create a list of one million random numbers with four blocks of code.

As a more complicated task, let's write a program that finds the largest item in the list. Now we need to start to think first and come up with an algorithm before we start coding. Think whether you can figure it out, before going to the next page…

Here is the basic idea: go through the list one by one and compare the current item to the max we found so far. If it is larger than the current max, we replace the current max with the current element, otherwise we do nothing. This works quite well, but there is one more important consideration: what should be the value of the currently found maximum when we start? If we know that all the number in the list are positive, for example, we can set that value to zero. That ensures that we will find a greater value at least once and hence, find the correct max. But what if there is no lower limit? How can we make sure that we will pick an element from the list when we do the comparison? For example, if the list can have negative values too, zero would be a bad choice. Why? What if all the elements are negative, then zero is greater than all of them, so our algorithm will find the wrong value, zero.

The solution is to pick one item of the list as the initial max. Typically, you want to use the first item, but it does not really matter which one. If you do this, then there are two cases: 1) either this item is the max and then we'll find the correct value since the algorithm will never change it or 2) there is at least one greater element and then our algorithm will find it.

Now, we are ready to turn our algorithm into code:

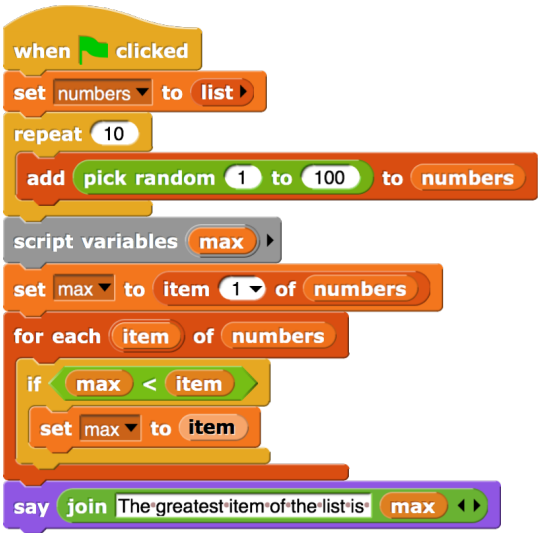


There are a few observations worth making:

- We use the **`length`** block to determine how many times the **`repeat`** loop should run since we want to look at every item.
- We need the **`index`** variable to be able to go through all possible index values. We start at 1 and we do need to **increment** its value at every iteration. It is a common mistake to forget it: if we do, we will look at the first element many times but not consider any other element!

Fortunately, there is a loop that was specifically designed for lists. To get it, we need to Import Tools using the File or Document menu. Once we do that, we'll get a bunch of extra custom blocks in the custom tab. There is a **`for`** loop that basically does what a repeat loop does but with a built-in index variable called i by default. Do try it out!

But there is an even better choice, the **`for each`** loop. Here is how we can simplify our maximum finding program with it:



The **`for each`** loop eliminates the need for an index variable. This loop simply iterates through the elements of a list: the variable **`item`** will take on each item one by one. The algorithm is the same as before, but the code is much simpler and cleaner.

Feel free to experiment with this program by creating a list of tens of thousands of random numbers and see how our program finds the maximum quickly. If you do that, however, remember to turn on Turbo Mode (settings/cogwheel menu). Otherwise, the program may take a while to complete.

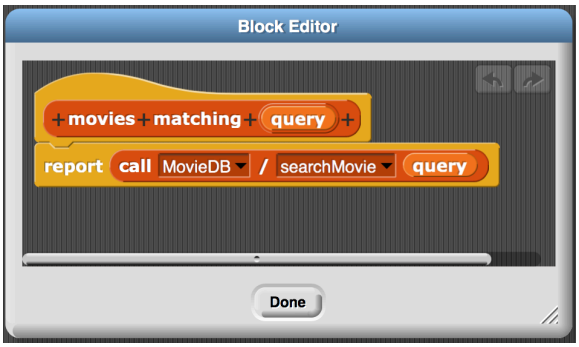
Now, we are ready to put lists into practical use by creating a Movie App.

The MovieDB service

The MovieDB service provides functionality similar to that of IMDb. It has quite a few RPCs:

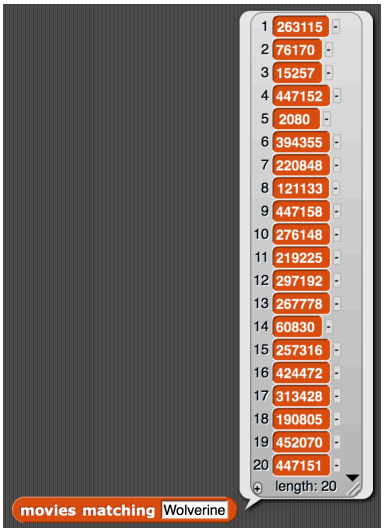
On smaller screens, you may not be able to see all of them. Fortunately, most services come with several predefined custom blocks. In the File Menu select Services and then from the list that pops up click Movie database. After a few seconds, the custom tab will have several new blocks:

Most of these are simply wrappers around RPC calls. For example, if you right click on the `movies matching` block and select Edit, this is what you'll see:



It is up to you whether you prefer these custom blocks or calling the RPCs directly.

Let's see what happens when we actually use one of these:



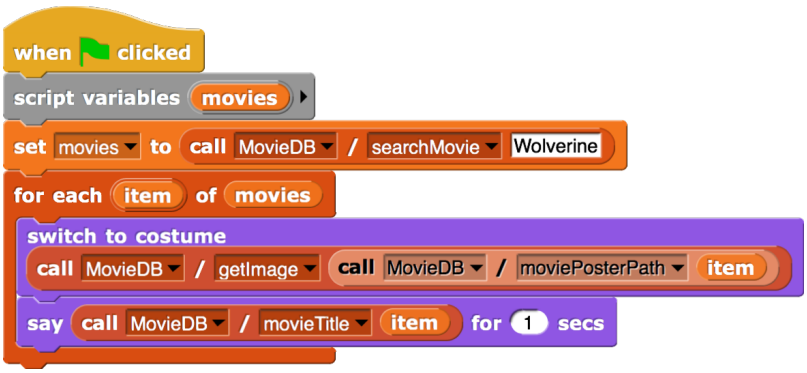
What is this list of numbers? Each movie in the database is associated with a unique ID and that is what you see here. The RPC returns the IDs of all the movies whose title contains the word Wolverine (to be more precise, the first 20 if there are more than 20).

To make use of it, let's see what the title of the first movie on that list:

First we call the **`movies matching`** custom block (that, in turn, calls the appropriate RPC), then we take the first item of the reported list and then use that movie ID as an input to the **`movieTitle`** RPC which returns the title, Logan.

The Movie App

Now we are ready to write an actual program about movies: Let's cycle threw all the Wolverine movies and show their posters:

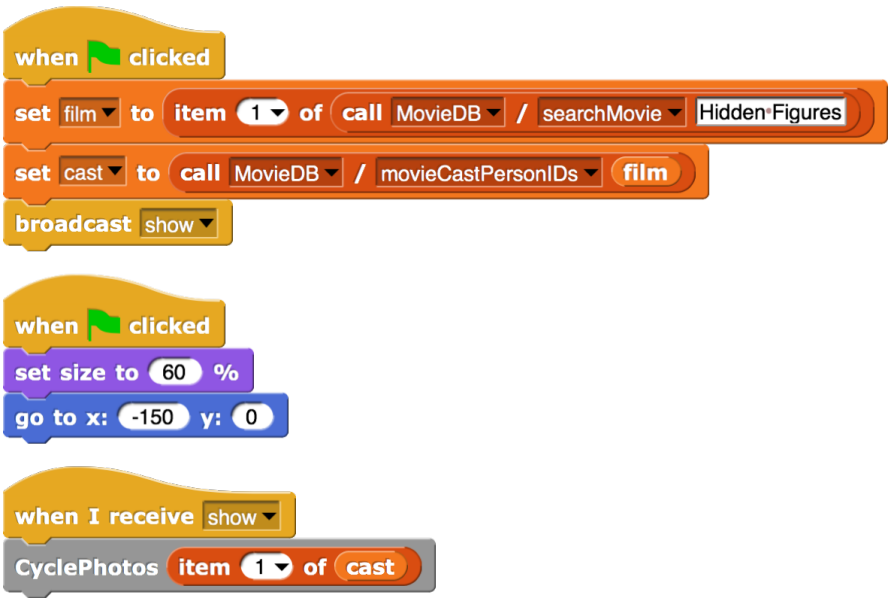


First, we get all the movies in a list and then we use the **`for each`** loop that is the best to visit all items in a list (you need to use the File menu Import Tools to get the **`for each`** loop under the Custom tab.) . There are two new RPCs we use here: the **`moviePosterPath`** will return a URL that identifies the image file in the database. So we use it as an input to the **`getImage`** RPC to get the actual image and switch the costume to it.

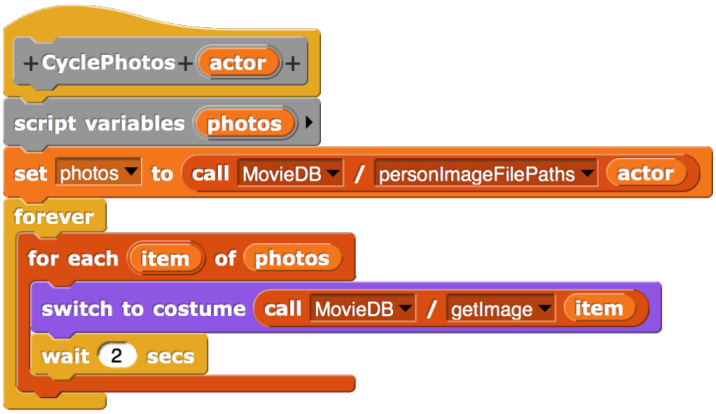
Since some movies do not have posters, we display the movie titles also. Here is a link to the program (with some minor differences, see whether you can spot them…):

https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=MoviePosters

A slightly more complicated program displays all the photos of the three leading cast members of a movie. It has three sprites; each is responsible for displaying photos of one cast member. Here are the scripts of the first one:

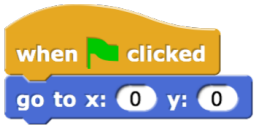


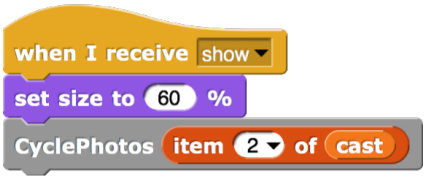
When the green flag is clicked, we get the first item of the list of movies matching “Hidden Figures.” Then we use the **`movieCastPersonIDs`** RPC to get a list of cast member IDs. Then we broadcast a new event signaling that we are ready to cycle through all the photos. We also set the position and size of the sprite. The **`CyclePhotos`** custom block is shown below:



It takes a person ID as an input (`actor` variable inside the block). Then we use the `personImageFilePaths` which will return a list of URLs to all available photos of the given person in the database. Then we'll cycle through all the pictures, leaving each up for 2 seconds. Since we do this in a `forever` loop, this will go on until we press the red stop sign.

The other two sprites are very similar, but even simpler. The second one is in the center, so its position is set to 0 0.





When the show event occurs, it simply calls the `CyclePhotos` custom block with the second element of the cast list. Check back on the previous page and see how the first sprite used the first element of the cast. Similarly, the third sprite (not shown) has a different position and uses the third item form the cast list.

This is a perfect illustration why custom blocks are very useful: we used the same one in the three sprites; each time with a different cast member.

Here is the program:
https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=CastShow

For a fun movie guessing game, try this project:
https://editor.netsblox.org/?action=example&ProjectName=Movies