```
/*  --------------------------------
        Lets Play With Array
    --------------------------------
    A simple MaxScript tutorial (by Anubis) [2010-03-04]
    written to help on question posted into the ScriptSpot forum.

    Of course, I will write it in manner what allow to read it without
    bound the context to the forum topic (to facilitate readers).
    Also I'll try to keep it as simple as possible for the novice coders.

    Interpreted question was is there a way to write array of objects
    in compact syntax like: #(Box01 to Box09, Box12, Box22 to Box39)
    instead of full: #(Box01, Box02, Box03, Box04, Box05, ... Box39)

    Briefly, nope :-)
    But MaxScript is a language as any other (including humans languages)
    and we can combine available technique to build what we need.

    The BitArray permit compact syntax like: #{1..9, 12, 22..39}
    Ok, what we need to know about them is that they can contain only
    integer values and the values must be positive (greater than zero).

    The standard Array can hold different type of data. In your case we
    need an object sets (array of nodes) and to filter out needless objects.
    And we can do that using BitArray as index holder. ;-)

    But, first, we need to keep attention to the "collecting" methods,
    because we wish to exclude objects by their numeric suffix.
    For example - we have 3 boxes with names Box01, Box02, Box03,
    and we intended to exclude Box01. At the glance looks simple:
*/
boxes = $Box* as array -- collect 'em
deleteItem boxes 1 -- remove item 1 from the array

/*
    Everything is fine if item 1 is actual your Box01 object. But not
    any guarantee items will be collected in order sorted by name.
    After cloning, deleting objects and so on the "order" is rearranged.
    Of course, using Select By Name dialog we can sort and select and
    then collect our selection...
*/
sel = selection as array

/*
    ...but if we intended to write a automatized script that will work
    correct regardless of user collecting activity, well, we need to sort
    our objects by name.

    Note: the sort() function need array items to be comparable.
    The objects itself are not comparable, but we can (and need)
    to sort by their names. So we need to collect them.

    Note: To make our variables and arrays temporary is good to insert
    them into local variable scope. This way they'll auto cleaned up then
    the scope execution end.

    Note: Create at least 39 boxes to can test the examples below.
*/
```

```
( -- example scope
   allBoxes = $Box* as array -- collect the objects
   objNames = for i in allBoxes collect i.name -- collect their names
   sort objNames -- sort them
   -- and finally using getNodeByName() collect them in "order"
   sortedObj = for i in objNames collect (getNodeByName i)
   print sortedObj -- see result into the Listener
)
/*
   As you see, I used 3 arrays, but just to make the example clear.
   In practice, even in a local scope, too many arrays will grab unduly
   amount of memory. At the beginning, above coding style is helpful,
   more clear, right? But in the future omit (forget it).
*/
(-- the same procedure using one array
   objs = for i in $Box* collect i.name -- get the names
   sort objs -- sort
   -- and collect back to the same array
   objs = for i in objs collect (getNodeByName i)
   print objs -- see result into the Listener
)

/*
   This was the first step, and we are ready for the next step.
   As for example our wanted objects are:
   (Box01 to Box09, Box12, Box22 to Box39), then...
*/
(
   -- we will use BitArray to define their index
   index = #{1..9, 12, 22..39}
   -- final step - redefine your array using that indexes
   objs = for i in index collect (objs[i])
   print objs -- see result into the Listener
)

/*
   Ok, it's normal to see the error message here :-)
   This is purposely to show the importance of variable scope context.
   Here our local "objs" array is undefined. So lets put all in one...
*/
(
   objs = for i in $Box* collect i.name
   sort objs
   objs = for i in objs collect (getNodeByName i)
   index = #{1..9, 12, 22..39}
   objs = for i in index collect (objs[i])
   select objs -- select them (for example)
)

/*
   That's all for now, I hope you enjoied my simple tutorial.
   If God willing, some day I'll write more :-)

      Best Regards,

      Anubis
      anubiss@abv.bg
*/
```