Subject: Generalisation of the use of lists in the IDL language
Posted by Fabzi on Fri, 16 May 2014 15:09:59 GMT
View Forum Message <> Reply to Message

Folks,

I really appreciate the recent efforts from IDL to become more modern (should I say "python-like"?) which made programming and data-exploring much more fun. Lists, ordered-hashes and dictionaries (which made hashes obsolete from my point of view) have replaced arrays, pointers and structures in much of my recent code.

However, in addition to the performance problems raised by Tom Grydeland, I also noticed that while my Input&Analysis workflow was made more "elegant" thanks to the new features, it became a bit more tedious when I arrived to the "output" (i.e. plotting or writing). Unlike NumPy, where "everything is a list", IDL built-in routines still require arrays as parameters.

I'll make an example:

```
IDL> data = list(indgen(10), /EXTRACT)
IDL> print, data[2:4] ; behaves like an array (almost)
       2
       3
       4
IDL> data[2:4] = data[4:6] ; also like an array
IDL> data[2:4] = data[2:4]^2 ; this won't work
% Unable to convert variable to type object reference.
% Execution halted at: $MAIN$

IDL> p = plot(data[2:4]) ; this won't work either
% PLOT: REFORM: New subscripts must not change the number elements in
<OBJREF    Array[1]>.
% Execution halted at: $MAIN$

IDL> p = plot(data[2:4]->toArray()) ; the ->toArray() is necessary
```

Mathematical operations on lists are not possible. I guess this wont change because the "+" operator is already overloaded and "adds" the elements to the list.

However, I was wondering if it would be theoretically possible that the built-in routines like mean(), plot() etc. also accept lists as arguments. I guess that the syntax would allow it, but at what costs?

I've already programmed several wrappers for IDL routines which made my life easier (e.g. wrappers for read_csv and write_csv now accepting

dictionaries) and I know that others did (histogram returning lists from Paulo: http://www.ppenteado.net/idl/histogram_pp-code.html)

Any thoughts about this, or is asking for more integration of lists an impossible idea?

Fabien

Subject: Re: Generalisation of the use of lists in the IDL language
Posted by kagoldberg on Fri, 23 May 2014 04:42:56 GMT
View Forum Message <> Reply to Message

I understand that what you're discussing here seems to be the limitations of lists, but I would suggest that you may not be using the right tool for the job. If you've got a list of scalar values, go ahead and use an array. The list doesn't give you any advantage. Lists are like pointer arrays. They're cumbersome for tasks that arrays perform natively, but powerful for mixed datasets, or individual large elements.

Use lists for more complex or varied information. You can have a list of structures, where not every structure is the same. Or a list of images, where each image can be a different size. Those are things you can't easily do with arrays. In the past, you could make a pointer array, but lists just made that a whole lot easier, because of the lack of pointer dereferencing, and the indexing.

Personally, I like the feature where the list's indices can seamlessly burrow into the individual elements. Like this:

```
IDL> a = list([10,11,12,13], [100,101,102,103], ['a','b','c','d'], ['Bob','Tom'])
IDL> print, a[3,1]
Tom
IDL> print, a[1,2]
    102
```

I don't know that this configuration would ever occur quite like this in a program I write, but that's a flexible way to retrieve information. It might be too much to ask to start performing all of the varied array operations on the elements of such a list.
