Subject: Re: HASH -- bug, or "feature"?
Posted by Michael Galloy on Wed, 20 Apr 2011 19:19:23 GMT
View Forum Message <> Reply to Message

On 4/20/11 1:12 PM, Gray wrote:
> Can anyone tell me why, when you create a hash, the elements are not
> in the order of the keys you give? And, even worse, when you index a
> hash with an array of keys, the resulting hash does not come out in
> the same order as the array?
>
> IDL> h = hash(['a','b','c','d'],indgen(4))
> IDL> print, h
> c: 2
> a: 0
> b: 1
> d: 3
> IDL> print, h[['a','c','d']]
> c: 2
> a: 0
> d: 3
>
> This is making my bookkeeping using hashes basically impossible.
>
> --Gray

Hashes are not an ordered collection.

If I wanted to store something in a hash, but order was important, I would use a list and a hash where the keys are added in the correct order to the list and then looked up in the hash to get the value.

Mike
--
www.michaelgalloy.com
Research Mathematician
Tech-X Corporation

Subject: Re: HASH -- bug, or "feature"?
Posted by Paul Van Delst[1] on Wed, 20 Apr 2011 19:50:59 GMT
View Forum Message <> Reply to Message

It's not a bug or a feature. It's a property of hashes. And it's not peculiar to IDL. E.g. from the Ruby pickaxe:

<quote>
Compared with arrays, hashes have one significant advantage: they can use any object as an index. However, they also

have a significant disadvantage: their elements are not ordered.
</quote>

Similarly with Python dictionaries.

Typically the need for hashes where insertion order is important leads to a subclass of Hash being created where the
order is internally tracked.

cheers,

paulv

Gray wrote:
> Can anyone tell me why, when you create a hash, the elements are not
> in the order of the keys you give? And, even worse, when you index a
> hash with an array of keys, the resulting hash does not come out in
> the same order as the array?
>
> IDL> h = hash(['a','b','c','d'],indgen(4))
> IDL> print, h
> c: 2
> a: 0
> b: 1
> d: 3
> IDL> print, h[['a','c','d']]
> c: 2
> a: 0
> d: 3
>
> This is making my bookkeeping using hashes basically impossible.
>
> --Gray

Subject: Re: HASH -- bug, or "feature"?
Posted by Mark Piper on Wed, 20 Apr 2011 20:33:11 GMT
View Forum Message <> Reply to Message

Here's an example of the technique Mike suggests:

IDL> names = list('red', 'green', 'blue')
IDL> colors = hash()
IDL> colors[names[0]] = [255,0,0]
IDL> colors[names[1]] = [0,255,0]
IDL> colors[names[2]] = [0,0,255]

```
IDL> print, colors ; order is mangled
green:       0     255       0
blue:        0       0     255
red:       255       0       0

IDL> foreach n, names do print, n, colors[n] ; ordered
red     255       0       0
green       0     255       0
blue        0       0     255
```

This also works with an array (and a FOR loop) instead of a list.

mp

Subject: Re: HASH -- bug, or "feature"?
Posted by Gray on Wed, 20 Apr 2011 21:26:14 GMT
View Forum Message <> Reply to Message

On Apr 20, 4:33 pm, Mark Piper <mpi...@ittvis.com> wrote:
> Here's an example of the technique Mike suggests:
>
> IDL> names = list('red', 'green', 'blue')
> IDL> colors = hash()
> IDL> colors[names[0]] = [255,0,0]
> IDL> colors[names[1]] = [0,255,0]
> IDL> colors[names[2]] = [0,0,255]
>
> IDL> print, colors ; order is mangled
> green: 0   255   0
> blue: 0   0   255
> red: 255   0   0
>
> IDL> foreach n, names do print, n, colors[n] ; ordered
> red   255   0   0
> green   0   255   0
> blue   0   0   255
>
> This also works with an array (and a FOR loop) instead of a list.
>
> mp

This all makes perfect sense... except that it isn't really useful for me. I had been using a hash so that I could retrieve and store information (in the form of structures) about particular stars by indexing with star IDs and not having to search over arrays or lists for individual members. When I had information for a set of stars

where some but not all were already in the hash, I would do something
like this:

tmp = replicate({star},n_new)
old = where(star_hash.haskey(new_ids),n_old)
if (n_old gt 0) then tmp[old] =
(star_hash[new_ids[old]].values()).toarray()
tmp.info = new_info & tmp.id = new_ids
star_hash[new_ids] = tmp

But that doesn't work, because there's no guarantee that the values I
retrieve from the hash are in the same order as the array elements I'm
storing them in. What I end up having to do is

tmp = replicate({star},n_new)
old = where(star_hash.haskey(new_ids),n_old)
foreach i,old do tmp[i] = star_hash[new_ids[i]]

...and the rest is the same. I'm not sure whether this is more
inefficient or not... maybe it isn't, thanks to toarray() being slow.

Subject: Re: HASH -- bug, or "feature"?
Posted by penteado on Wed, 20 Apr 2011 22:09:42 GMT
View Forum Message <> Reply to Message

On Apr 20, 6:26 pm, Gray <grayliketheco...@gmail.com> wrote:
> This all makes perfect sense... except that it isn't really useful for
> me. I had been using a hash so that I could retrieve and store
> information (in the form of structures) about particular stars by
> indexing with star IDs and not having to search over arrays or lists
> for individual members. When I had information for a set of stars
> where some but not all were already in the hash, I would do something
> like this:
>
> tmp = replicate({star},n_new)
> old = where(star_hash.haskey(new_ids),n_old)
> if (n_old gt 0) then tmp[old] =
> (star_hash[new_ids[old]].values()).toarray()
> tmp.info = new_info & tmp.id = new_ids
> star_hash[new_ids] = tmp

I have a subclass for ordered hashes, which I could clean up and make
public if there is interest. However, I do not see why it is needed
above. If I understand it right, you want to put the new elements in
the hash, without overwriting any preexisting elements. Would it not
be the same as just

tmp=replicate({star},n_new)
tmp.info=new_info
tmp.id=new_ids
new=where(~star_hash.haskey(new_ids),/null)
star_hash[new_ids[new]]=tmp[new]

?

The work being just to avoid overwriting the preexisting elements. If
they could be overwritten, it would be just

tmp=replicate({star},n_new)
tmp.info=new_info
tmp.id=new_ids
star_hash[new_ids]=tmp

Subject: Re: HASH -- bug, or "feature"?
Posted by Gray on Thu, 21 Apr 2011 11:20:27 GMT
View Forum Message <> Reply to Message

On Apr 20, 6:09 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Apr 20, 6:26 pm, Gray <graylikethec...@gmail.com> wrote:
>
>> This all makes perfect sense... except that it isn't really useful for
>> me. I had been using a hash so that I could retrieve and store
>> information (in the form of structures) about particular stars by
>> indexing with star IDs and not having to search over arrays or lists
>> for individual members. When I had information for a set of stars
>> where some but not all were already in the hash, I would do something
>> like this:
>
>> tmp = replicate({star},n_new)
>> old = where(star_hash.haskey(new_ids),n_old)
>> if (n_old gt 0) then tmp[old] =
>> (star_hash[new_ids[old]].values()).toarray()
>> tmp.info = new_info & tmp.id = new_ids
>> star_hash[new_ids] = tmp
>
> I have a subclass for ordered hashes, which I could clean up and make
> public if there is interest. However, I do not see why it is needed
> above. If I understand it right, you want to put the new elements in
> the hash, without overwriting any preexisting elements. Would it not
> be the same as just
>
> tmp=replicate({star},n_new)
> tmp.info=new_info
> tmp.id=new_ids

> new=where(~star_hash.haskey(new_ids),/null)
> star_hash[new_ids[new]]=tmp[new]
>
> ?
>
> The work being just to avoid overwriting the preexisting elements. If
> they could be overwritten, it would be just
>
> tmp=replicate({star},n_new)
> tmp.info=new_info
> tmp.id=new_ids
> star_hash[new_ids]=tmp

It's worse than that. tmp.info = new_info was shorthand for updating the relevant structure tags with the new information; however, there are other tags with old information that I don't want to overwrite.
So I need to preserve some stuff and overwrite others, which is why I had to do the complicated jig.
