Ruby master - Feature #12902

How about Enumerable#sum uses initial value rather than 0 as default?

11/04/2016 07:24 PM - AaronLasseigne (Aaron Lasseigne)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

In https://bugs.ruby-lang.org/issues/12217#note-3, Akira Tanaka, mentions that the default argument to sum is 0. This creates problems with non-numeric summations (e.g. strings). This would make the method more flexible. It also makes it behave more like reduce. I think using the initial value in the enumerable is less surprising than using 0.

History

#1 - 11/04/2016 07:28 PM - AaronLasseigne (Aaron Lasseigne)

In https://bugs.ruby-lang.org/issues/12217#note-3, Akira Tanaka, mentions that the default argument to sum is 0. This creates problems with non-numeric summations (e.g. strings). I would like to suggest using the first enumerable value. This would make the method more flexible. It also makes it behave more like reduce. I think using the initial value in the enumerable is less surprising than using 0.

** I missed a sentence. :(**

#2 - 11/05/2016 02:29 AM - akr (Akira Tanaka)

- Status changed from Open to Feedback

I think [].sum should return 0.
Returning 0 is better than nil because programmers don't need to treat empty array specially.
This behavior is different with Enumerable#reduce which returns nil for empty array.

Also, string summation using Enumerable#sum is not good idea because it is slow.
String#join is faster.
So, we should not encourage Enumerable#sum for string summation.

#3 - 11/05/2016 04:01 AM - duerst (Martin Dürst)

On 2016/11/05 04:28, aaron.lasseigne@gmail.com wrote:

> Issue #12902 has been updated by Aaron Lasseigne.
>
> In https://bugs.ruby-lang.org/issues/12217#note-3, Akira Tanaka, mentions that the default argument to sum is 0. This creates problems with non-numeric summations (e.g. strings). I would like to suggest using the first enumerable value. This would make the method more flexible. It also makes it behave more like reduce. I think using the initial value in the enumerable is less surprising than using 0.

What would/should the behavior be for an empty array?

#4 - 11/05/2016 02:35 PM - herwin (Herwin W)

Akira Tanaka wrote:

> I think [].sum should return 0.
> Returning 0 is better than nil because programmers don't need to treat empty array specially.
> This behavior is different with Enumerable#reduce which returns nil for empty array.
>
> Also, string summation using Enumerable#sum is not good idea because it is slow.
> String#join is faster.
> So, we should not encourage Enumerable#sum for string summation.

I guess you mean Array#join

And the fact that Enumerable#sum could be used for string concatenation makes the default value of 0 pretty weird. Just imagine the following piece of code:

```
a = ['']
a.sum    => ''
a.delete('')
```

```
a.sum    => 0
```

This just removes the special case of empty arrays for numeric values, but still doesn't help much with anything else (even though I agree that calculating a sum of strings is just stupid)

#5 - 11/11/2016 08:31 PM - AaronLasseigne (Aaron Lasseigne)

Martin,

I would expect the empty to act like reduce and return nil.

Akira,

I agree that doing string concatenation with sum is a poor idea. It's just that this is on Enumerable which is supposed to allow iterations over whatever underlying data you want and this is favoring a particular type of underlying data. I do understand that 0 is convenient for the vast majority of use cases, it just feels like Numeric is creeping into Enumerable.

Thanks for considering it and responding.

#6 - 11/11/2016 10:16 PM - bitsweat (Jeremy Daer)

If we treated the first argument to #sum as the additive identity instead of "initial element," then it'd be very clear what to expect:

When there are no elements in the Enumerable, return the identity.

That works nicely for summing non-numeric/string/etc objects:

```
payments.none? # => true
payments.sum(Payment.new(0)) # => Payment.new(0)
```

Plus, it's backward compatible, doesn't break existing behavior.

#7 - 11/12/2016 12:10 AM - akr (Akira Tanaka)

- Status changed from Feedback to Rejected

Jeremy Daer wrote:

> If we treated the first argument to #sum as the additive identity instead of "initial element," then it'd be very clear what to expect:

I like current behavior than yours because the behavior can be explained simpler:
Enumerable#sum returns initial + elem0 + elem1 + ... elemN.
No need to distinguish empty and non-empty enumerable.

Yours needs more complex explanation.

> Plus, it's backward compatible, doesn't break existing behavior.

Enumerable#sum and Array#sum is defined in many gems with different behaviors.
Perfect backward compatibility is impossible.

#8 - 12/22/2016 01:40 AM - ioquatix (Samuel Williams)

Just to add, this breaks code which uses facets.

https://github.com/rubyworks/facets/issues/247

The method signature is:

```
def sum(*identity, &block)
```

I think it's probably important to consider this use case.