archery Documentation

Release 0.1.0

Julien HawkeEye Tayon

May 31, 2012

CONTENTS

- Source : https://github.com/jul/archery
- Tickets : https://github.com/jul/archery/issues?state=open
- Latest documentation : http://archery.readthedocs.org/en/latest/index.html

It is an enhancement of MutableMapping based on Mixins. It currently only offers:

- addition;
- substraction;
- multiplication;
- division.

Last but no least, it is fun.

CHAPTER

ONE

DETAILED DOCUMENTATION

Contents:

1.1 Package content

1.1.1 archery.trait

see: *Traits are Mixins*

Traits are the Perl equivalent of mixins (ruby behaviour). The mixins in archery are dedicated to provide customisable operator on mapping for :

- addition,
- substraction,
- division,
- multiplication

archery.trait provides individual trait.

1.1.2 archery.quiver

see *Quivers : consistent sets of Traits*

A quiver is a set of traits lovingly assembled so that they are consistent.

1.1.3 archery.bow

see *Bow: how not to shoot yourself an arrow in the knee*

Ready made MutableMapping (dict) that supports addition.

1.1.4 archery.barrack

see *barrack*

Misc utilities.

1.2 Traits are Mixins

Traits are Mixins, behaviours. All these terms recovers loosely the same idea.

In this case refering to even older conventions traits are concrete classes for abstract classes/interfaces.

collections.MutableMapping defines an interface and some concrete methods. Since isinstance relies on interfaces (ducktyping) I can safely use it to implement methods that don't exists and will normally work for most Mappings.

1.2.1 General Rules:

Generic behaviour

The Operation is propagated for each keys of both Mappings and will be propagated to the ending values. If the values are MutableMapping with the operation they will propagate. If the values are not MutableMapping with the trait, the operation will apply in place.

> **Warning:** If your MutableMapping with Addition is made of MutableMapping without it, you'll have a problem. To solve the problem use **:ref:'bowyer'_**

Scalar Operations

A scalar is everything that is not a MutableMapping. Trait support things such as integer, array operation by applying the operation on each values of the MutableMapping. Order is respected.

Inclusive Trait

If a key is absent on one of the Mapping, it will be considered the neutral element. An empty list, for list, 0 for int, 0.0 for float...

The behaviour of addition and substraction is consistently deriving from the boolean algebrae meaning of **+** in a set context where + means union.

Thus Addition and substraction are inclusive.

Exclusive Trait

Multiplication operates as an intersection, because on one hand it is consistentwith the set/boolean meaning of multiplication, and also that neutral element of addition, is normaly the null element of multiplication. Since multiplication implies division, instead of multiplying by 0 and keeping present in at least one dict, I prefer to avoid the raging division by zero. In short, I try to avoid my dict to explode when dividing by 0. I am weak I know.

1.2.2 Summary of the behaviours and dependancies

Operation	Short	Behaviour	Requires	Safe	Name
Copier	copy	None			
Addition	add	Inclusive	copy	Yes	InclusiveAdder
Multiplication	mul	Exclusive	add,copy	Yes	ExclusiveMuler
Substraction	sub	Inclusive	add,mul,copy	Yes	InclusiveSubber
Division	div	Exclusive	add,mul,sub,copy	No	TaintedExclusiveDiver

1.2.3 Caveat

Why is dividing unsafe?

http://beauty-of-imagination.blogspot.fr/2012/05/dividing-is-not-as-easy-at-it-seems.html

1.3 Quivers : consistent sets of Traits

Yes, it is a pun, trait = arrow <=> quiver = set of arrows.

1.3.1 Why quivers?

My purpose is not to make trait for the sake of making traits. It is to have a bigger set of concrete class for purpose. But I want them to be consistent.

My (not yet available) unnitest for trait check for actual results (such as 2+2=4), my unnitest (not yet available) test for how well the operations are coupled in terms of commutation, symmetry, distributivity, associativity.

I see nothing wrong in changing the behviour of add and sub (you may want to have fun with non euclidian space). But you may want an algebra to follow the least surprise principle. That's all about it.

1.3.2 Available quivers

SimplyAdd

A quiver to make your MutableMapping add:

```
>>> from archery.quiver import SimplyAdd
>>> class Daddy(SimplyAdd, dict): pass
>>>
>>> a= Daddy({'x' : 1 , 'y' : []}, 'z' : "hell")
>>> b= Daddy({'x' : 2 , 'y' : [1,3], 'z', "o"})
>>> print a+b
# {'y': [1, 3], 'x': 3, 'z': 'hello'}
>>> a+=b
>>> print a
# {'y': [1, 3], 'x': 3, 'z': 'hello'}
>>>
>>> r=Daddy(a=1)
>>> print r+1
# {'a': 2}
>>> print -1+a
# {'a':0}
```

If you are smart, you almost have the subtraction ^_^

LinearAlgebrae

A quiver to make your MutableMapping support + - / *

Pretty much as easy to use as SimplyAdd

1.4 Bow: how not to shoot yourself an arrow in the knee

The only reason to use a bow is either you are lazy, or you noticed I only test these. This is the most tested part of the code yet.

My philosophy in testing is: check the component relying on the maximum of your code gives satisfaction since I should traverse most of the point of failure, and then I add a unittest per bug found.

1.4.1 Dankyu

A dict with addition used here : http://github.com/jul/parseweblog

It instanciate like a dict::

```
>>> from archery.bow import Hankyu
>>> a = Hankyu(x=1,y=2,z=2)
```

It adds :) ::

```
>>> a+a
{'y': 4, 'x': 2, 'z': 4}
>>> a+-2
{'y': 0, 'x': -1, 'z': 0}
>>> a+.5
{'y': 2.5, 'x': 1.5, 'z': 2.5}
>>> .5+a
{'y': 2.5, 'x': 1.5, 'z': 2.5}
```

It adds everything consistent with your value:

```
>>> a = Hankyu(data=[1,2],sample=1)
>>> b = Hankyu(data=[3,4],sample=1)
>>> a+b
{'sample': 2, 'data': [1, 2, 3, 4]}
```

If propagates the addition to the included bow::

```
>>> a=Hankyu(a=Hankyu(a=1))
>>> b=Hankyu(a=Hankyu(a=2))
>>> a+b
{'a': {'a': 3}}
>>> a+=1
>>> a
{'a': {'a': 2}}
>>>
```

1.4.2 Daikyu

A dict that loves to do a lot of things : * addition; * substraction; * multiplication; * division (please, please be careful).

Simple algebrae

It instanciates like a dict:

```
>>> from archery.bow import Daikyu
>>> b=Daikyu(x=2, z=-1)
>>> a=Daikyu(x=1, y=2.0)
>>> a+b
# OUT: {'y': 2.0, 'x': 3, 'z': -1}
>>> b-a
# OUT: {'y': -2.0, 'x': 1, 'z': -1}
>>> -(a-b)
# OUT: {'y': -2.0, 'x': 1, 'z': -1}
>>> a+1
# OUT: {'y': 3.0, 'x': 2}
>>> -1-a
>>> # OUT: {'y': -3.0, 'x': -2}
>>> a*b
# OUT: {'x': 2}
>>> a/b
# OUT: {'x': 0}
>>> 1.0*a/b
# OUT: {'x': 0.5}
```

Why div is special?

Because div is special and I stick to python 2 behaviour on this one.

http://beauty-of-imagination.blogspot.fr/2012/05/dividing-is-not-as-easy-at-it-seems.html

Don't flame me yet, I can provide another diver, but my brain is yet kaput.

See by yourself::

```
>>> b/2
# OUT: {'x': 0, 'z': 0}
>>> b/2.0
# OUT: {'x': 1.0, 'z': -0.5}
>>> 2/b
# OUT: {'x': 0, 'z': -2}
```

But you can correct this::

```
>>> 2.0/(1.0*b)
# OUT: {'x': 1.0, 'z': -2.0}
```

Mixing scalars and records

My prefered part :) ::

```
>>> 2*Daikyu(x=1, y="lo",z=[2])
{'y': 'lolo', 'x': 2, 'z': [2, 2]}
>>> Daikyu(y=1, z=1)*Daikyu(x=1, y="lo",z=[2])*2
{'y': 'lolo', 'z': [2, 2]}
>>> a=Daikyu(dictception=Daikyu(a=1,b=2), sample = 1, data=[1,2])
>>> b=Daikyu(dictception=Daikyu(c=-1,b=2), sample = 2, data=[-1,-2])
>>> a+b
{'sample': 3, 'dictception': {'a': 1, 'c': -1, 'b': 4}, 'data': [1, 2, -1, -2]}
>>> Daikyu(dictception=1, sample=1)* a*b
{'sample': 2, 'dictception': {'b': 4}}
```

1.5 Misc interesting questions

1.5.1 What is addition in MutableMapping useful for?

It is used with parseweblog as an exemple. I find addition on MutableMapping a very convenient way to reduce by using in place addition (__iadd__).

VectorDict also has an exemple of map/reduce with multiprocessing word counting

MapReduce is a way of treating big data without consuming too much memory ensuring relativley good performance. It is normaly considered to belong to the functional paradigm and is best used with generators.

1.5.2 Doesn't it overlaps with defaultdict?

No. Results seems similar, but the collections.defaultdict philosophy is different and does not mix in very well with archery because you have a conflict. I therefore admit, there is a design flaw in VectorDict.

defaultdict creates missing key from a factory (a function with void argument) and will consider that asking for a key that does not exists makes it real.

MutableMapping with the default traits will raise an Exception in such case, however, if you add to MutableMapping, as the default Adder is Inclusive, it will add the exisiting key of the source and destination. No values in MutableMapping with the default Adder will exists unless there are already defined in the MutableMappings.

Since traits are flexible, You or I could provide more stricts dict.

Trait seems to provide autovivification but it does not ! No values will be created on the fly.

As a result, there is a conflict between defaultdict and traits : for instance with a defaultdict when you add with a value that does not exists in one of the dict you should use the default factory. With actual traits, it is assumed the value is the neutral element of addition, thus having far less problems than with defaultdict.

1.5.3 Why so much fuss on Algebrae if you use Addition 99% of the time?

Because Algebrae is not about knowing the value of 1 + 1, it is about consistency rules for operator. People usually focus on the operand of an operation to check if it works, I focus on the operator behaviour and how well they behave together. Mathematical symbols are a litterature whose intuition can safely work if we stay in the safeguard of the acceptable behaviour. These behaviours are commonly refered: distributivity, neutral element, scalar multiplication (or linear combinations), associativity.

Algebrae, is as a result for me only a functional test for the macro behaviour of addition. Addition alone has **strictly** no sense.

1.5.4 What is your naming convention, and what is archery exactly?

It is all explained her : http://beauty-of-imagination.blogspot.com/2012/05/joice-and-headache-of-naming.html

1.5.5 Why do you ask to sign a legal disclaimer with my blood and sign a pact with Satan if I want to use div?

We you look into the abyss, the abyss look into you: http://beauty-of-imagination.blogspot.fr/2012/05/dividing-is-not-as-easy-at-it-seems.html

Using division will make you lose your sanity and your confidence in computers reliability. Unless, you are fully prepared for this, and you have agreed I warned you, and you really know what you are doing : **I warn you to avoid using division on MutableMapping.**

I can use div since I don't fear losing what I am deprived of (sanity).

1.5.6 Who needs archery?

- people wanting to experiment what a good addition on any MutableMapping (dict included) coud be (**trait** documentation is for them);
- people wanting to have a consistent set of operations for their MutableMapping (**quiver** is for them);
- those who wants ready made dict pretty practical for map/reduce (**bow** is for them).

1.6 Changelog and roadmap

1.6.1 Convention:

version x.y.z

while in beta convention is :

- **x** = 0
- **y** = API change
- **z** = bugfix and/or improvement

and then

- **x** = API change
- **y** = improvement
- **z** = bugfix

1.6.2 Roadmap

Maybe backporting the search find and replace feature of VectorDict

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*