How to use numpy with 'None' value in Python?

I'd like to calculate the mean of an array in Python in this form:

```
Matrice = [1, 2, None]
```

I'd just like to have my `None` value ignored by the `numpy.mean` calculation but I can't figure out how to do it.

python numpy mean

edited May 3 '15 at 20:25
Alex Riley
53.5k 16 113 132

asked Jun 7 '09 at 17:21
clowny
555 5 10 21

2 +1: this question can be particularly relevant for arrays that are imported from a database, where values can sometimes be NULL. – EOL Nov 22 '11 at 22:30

5 Answers

You are looking for masked arrays. Here's an example.

```
import MA
a = MA.array([1, 2, None], mask = [0, 0, 1])
print "average =", MA.average(a)
```

Unfortunately, masked arrays aren't thoroughly supported in numpy, so you've got to look around to see what can and can't be done with them.

edited Aug 5 '15 at 22:21
demongolem
5,941 12 60 83

answered Jun 7 '09 at 18:10
tom10
38.3k 4 68 98

2 a member function that helped a lot was `filled` . that brought the masked array back to a normal array, filled with a value that I would recognize as invalid (NaN, -9999, whatever your users need). – mariotomo Apr 22 '10 at 9:20

Performance of masked arrays is also significantly less than regular numpy arrays as the implementation is pure Python. If you are dealing with big data, be aware of the performance implications. – timbo Dec 3 '14 at 23:37

You can use scipy for that:

```
import scipy.stats.stats as st
m=st.nanmean(vec)
```

answered Nov 22 '11 at 22:15
Noam Peled

Thanks, this is just what I needed! – max Jan 26 '12 at 9:20

1 This doesn't work. `a = [1,2,None]` and then `st.nanmean(a)` results in a TypeError. – Nate Jun 26 '13 at 20:58

1 Yes, you are right, it works on numpy.nan, not on None. It's most useful when calculating the mean on numpy vector. – Noam Peled Jun 30 '13 at 15:18

1 Now you can use also numpy.nanmean – Noam Peled Dec 11 '15 at 3:10

haven't used numpy, but in standard python you can filter out `None` using list comprehensions or the filter function

```
>>> [i for i in [1, 2, None] if i != None]
[1, 2]
>>> filter(lambda x: x != None, [1, 2, None])
[1, 2]
```

and then average the result to ignore the `None`

answered Jun 7 '09 at 17:28

cobbal
54k 12 119 146

4 `x != None` is usually written `x is not None` (PEP 8: "Comparisons to singletons like None should always be done with 'is' or 'is not', never the equality operators.") – EOL Nov 22 '11 at 22:27

You might also be able to kludge with values like NaN or Inf.

```
In [1]: array([1, 2, None])
Out[1]: array([1, 2, None], dtype=object)

In [2]: array([1, 2, NaN])
Out[2]: array([  1.,   2.,  NaN])
```

Actually, it might not even be a kludge. Wikipedia says:

> NaNs may be used to represent missing values in computations.

Actually, this doesn't work for the mean() function, though, so nevermind. :)

```
In [20]: mean([1, 2, NaN])
Out[20]: nan
```

edited Jun 16 '11 at 16:07

answered Dec 6 '09 at 2:26

endolith
8,064 14 67 134

6 Actually, `mean(a[~isnan(a)])` explicitly choosing all non-NaN values works. – u0b34a0f6ae Dec 7 '09 at 14:27

1 @kaizer your comment is a gem. great solution, thanks! – Agos Jun 16 '11 at 13:46

You can also use filter, pass None to it, it will filter non True objects, also 0, :D So, use it when you dont need 0 too.

```
>>> filter(None,[1, 2, None])
[1, 2]
```

answered Dec 6 '09 at 2:30

YOU
71.5k 18 135 197