**python how to summarize a list of tuples by first element**

<http://stackoverflow.com/questions/15722222/python-list-of-tuples-sum-second-val-of-each-tuple-only-if-first-val-of-tuple>

|  |
| --- |
| I have a list of "tagged" tuples...where each tuple is (tag\_id, value).like so:  my\_list = [(tag\_A, 100), (tag\_A, 200), (tag\_A, 300), (tag\_A, 400), (tag\_B, 400), (tag\_B, 600)]  I want to sum the values of each tuple with the same tag...so that:  sum\_of\_all\_values\_with\_tag\_A() = 1000  sum\_of\_all\_values\_with\_tag\_B() = 1000  I can't figure out a simple Pythonic way of doing that.  sum(set(value for tag\_id, value in my\_list))  returns the sum of ALL the values. |

|  |
| --- |
| **Approach**  Put your data into a defaultdict(list). Summarize that.  **Code**  from collections import defaultdict  my\_list = [('tag\_A', 100), ('tag\_A', 200), ('tag\_A', 300), ('tag\_A', 400), ('tag\_B', 400), ('tag\_B', 600)]  d = defaultdict(list)  for tag, num in my\_list:  d[tag].append(num)  **Test**  >>> from collections import defaultdict  >>> my\_list = [('tag\_A', 100), ('tag\_A', 200), ('tag\_A', 300), ('tag\_A', 400), ('tag\_B', 400), ('tag\_B', 600)]  >>>  >>> d = defaultdict(list)  >>> for tag, num in my\_list:  ... d[tag].append(num)  ...  >>> from pprint import pprint  >>> pprint(dict(d))  {'tag\_A': [100, 200, 300, 400], 'tag\_B': [400, 600]}  >>>  >>> pprint({k: sum(v) for k, v in d.iteritems()})  {'tag\_A': 1000, 'tag\_B': 1000}  **Alternative summary routine**  def summarize\_by\_tag(d):  for k, v in d.iteritems():  print k, sum(v)  >>> summarize\_by\_tag(d)  tag\_A 1000  tag\_B 1000 |

|  |  |
| --- | --- |
|  | As in other answers I would just use the defaultdict but unless you need the groups again later. Just sum them as you group. my\_list could then be a very large iterable and you're not storing the whole thing in memory.  from collections import defaultdict  my\_list = [('tag\_A', 100), ('tag\_A', 200), ('tag\_A', 300), ('tag\_A', 400), ('tag\_B', 400), ('tag\_B', 600)]  result = defaultdict(int)  for tag, value in my\_list:  result[tag] += value  print result  defaultdict(<type 'int'>, {'tag\_A': 1000, 'tag\_B': 1000}) |

Use a generator expression to sum per tag:

sum(val for tag, val in my\_list if tag == tag\_A)

You could *sort* on the tags then use itertools.groupby to create per-tag groups and sums:

from itertools import groupby

from operator import itemgetter

key = itemgetter(0) # tag

sums = {tag: sum(tup[1] for tup in group)

for tag, group in groupby(sorted(my\_list, key=key), key=key)}

This would produce a dictionary mapping tags to per-tag sum:

>>> from itertools import groupby

>>> from operator import itemgetter

>>> tag\_A, tag\_B = 'A', 'B'

>>> my\_list = [(tag\_A, 100), (tag\_A, 200), (tag\_A, 300), (tag\_A, 400), (tag\_B, 400), (tag\_B, 600)]

>>> key = itemgetter(0) # tag

>>> sums = {tag: sum(tup[1] for tup in group)

... for tag, group in groupby(sorted(my\_list, key=key), key=key)}

>>> print sums

{'A': 1000, 'B': 1000}

In your first solution, you can unpack tup into separate values in the for tup in part of the generator expression, which might make the other parts clearer. That is, sum(value for tag, value in my\_list if tag = tag\_A)

As in other answers I would just use the defaultdict but unless you need the groups again later. Just sum them as you group. my\_list could then be a very large iterable and you're not storing the whole thing in memory.

from collections import defaultdict

my\_list = [('tag\_A', 100), ('tag\_A', 200), ('tag\_A', 300), ('tag\_A', 400), ('tag\_B', 400), ('tag\_B', 600)]

result = defaultdict(int)

for tag, value in my\_list:

result[tag] += value

print result

defaultdict(<type 'int'>, {'tag\_A': 1000, 'tag\_B': 1000})

without importing anything. .

mysum={}

my\_list = [('tag\_A', 100), ('tag\_A', 200), ('tag\_A', 300), ('tag\_A', 400), ('tag\_B', 400), ('tag\_B', 600)]

for x in my\_list:

mysum.setdefault(x[0],0)

mysum[x[0]]+=x[1]

print mysum

output:

{'tag\_A': 1000, 'tag\_B': 1000}