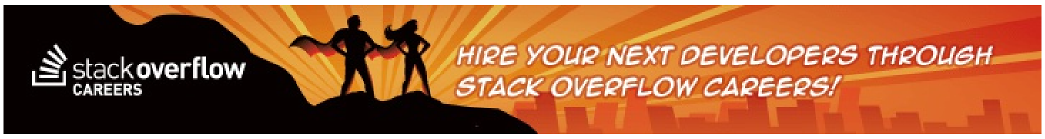
How do you return multiple values in Python?



The canonical way to return multiple values in languages that support it is often tupling.

Option: Using a tuple

Consider this trivial example:

```
def f(x):
  y0 = x + 1
  y1 = x * 3
  y2 = y0 ** y3
  return (y0,y1,y2)
```

However, this quickly gets problematic as the number of values returned increases. What if you want to return four or five values? Sure, you could keep tupling them, but it gets easy to forget which value is where. It's also rather ugly to unpack them wherever you want to receive them.

Option: Using a dictionary

The next logical step seems to be to introduce some sort of 'record notation'. In python, the obvious way to do this is by means of a `dict`.

Consider the following:

```
def g(x):
  y0 = x + 1
  y1 = x * 3
  y2 = y0 ** y3
  return {'y0':y0, 'y1':y1 ,'y2':y2 }
```

(edit- Just to be clear, y0, y1 and y2 are just meant as abstract identifiers. As pointed out, in practice you'd use meaningful identifiers)

Now, we have a mechanism whereby we can project out a particular member of the returned object. For example,

```
result['y0']
```

Option: Using a class

However, there is another option. We could instead return a specialized structure. I've framed this in the context of Python, but I'm sure it applies to other languages as well. Indeed, if you were working in C this might very well be your only option. Here goes:

```
class ReturnValue(object):
  def __init__(self, y0, y1, y2):
     self.y0 = y0
     self.y1 = y1
     self.y2 = y2

def g(x):
  y0 = x + 1
  y1 = x * 3
  y2 = y0 ** y3
  return ReturnValue(y0, y1, y2)
```

In python the previous two are perhaps very similar in terms of plumbing- After all `{ y0, y1, y2 }` just end up being entries in the internal `__dict__` of the `ReturnValue`.

There is one additional feature provided by Python though for tiny objects, the `__slots__` attribute. The class could be expressed as:

```
class ReturnValue(object):
  __slots__ = ["y0", "y1", "y2"]
  def __init__(y0, y1, y2):
     self.y0 = y0
     self.y1 = y1
     self.y2 = y2
```

From the Python Reference Manual:

> The `__slots__` declaration takes a sequence of instance variables and reserves just enough space in each instance to hold a value for each

The `__slots__` declaration takes a sequence of instance variables and reserves just enough space in each instance to hold a value for each variable. Space is saved because `__dict__` is not created for each instance.

Option: Using a list

Another suggestion which I'd overlooked comes from Bill the Lizard:

```
def h(x):
  result = [x + 1]
  result.append(x * 3)
  result.append(y0 ** y3)
  return result
```

This is my least favorite method though. I suppose I'm tainted by exposure to Haskell, but the idea of mixed-type lists has always felt uncomfortable to me. In this particular example the list is -not- mixed type, but it conceivably could be. A list used in this way really doesn't gain anything with respect to the tuple as far as I can tell. The only real difference between lists and tuples in Python is that lists are mutable, wheras tuples are not. I personally tend to carry over the conventions from functional programming: use lists for any number of elements of the same type, and tuples for a fixed number of elements of predetermined types.

Question

After the lengthy preamble, comes the inevitable question. Which method (do you think) is best?

I've typically found myself going the dictionary route because it involves less set-up work. From a types perspective however, you might be better off going the class route, since that may help you avoid confusing what a dictionary represents. On the other hand, there are some in the Python community that feel implied interfaces should be preferred to explicit interfaces, at which point the type of the object really isn't relevant, since you're basically relying on the convention that the same attribute will always have the same meaning.

So, how do -you- return multiple values in Python?

python coding-style return return-value

edited Sep 23 '15 at 10:55
jessag
382 ●4 ●17

asked Dec 10 '08 at 1:55
saffsd
4,937 ●7 ●40 ●57

1 In you excellent examples you use variable `y3`, but unless y3 is declared global, this would yield `NameError: global name 'y3' is not defined` perhaps just use `3`? – hetepeperfan Dec 19 '13 at 14:49

I think you missing a "self" in "def __init__(y0, y1, y2)". It should be "def __init__(self, y0, y1, y2)" – cgao Mar 10 '15 at 18:05

11 Answers

Named tuples were added in 2.6 for this purpose. Also see os.stat for a similar builtin example.

```
>>> import collections
>>> point = collections.namedtuple('Point', ['x', 'y'])
>>> p = point(1, y=2)
>>> p.x, p.y
1 2
>>> p[0], p[1]
1 2
```

edited Nov 19 '13 at 1:33
kindall
83.9k ●8 ●104 ●166

answered Dec 10 '08 at 16:36
Coady
17k ●4 ●18 ●26

12 This is only correct answer because it is the only canonical structure that the OP did not consider and because it addresses his problem of managing long tuples. Should be marked as accepted. – André Terra Jul 19 '14 at 22:35

Well, the design rationale for `namedtuple` is having a smaller memory footprint for *mass* results (long lists of tuples, such as results of DB queries). For individual items (if the function in question is not called often) dictionaries and classes are just fine as well. But namedtuples are a nice/nicer solution in this case as well. – Lutz Prechelt Feb 23 at 17:07

For small projects I find it easiest to work with tuples. When that gets too hard to manage (and not before) I start grouping things into logical structures, however I think your suggested use of dictionaries and ReturnValue objects is wrong (or too simplistic).

Returning a dictionary with keys y0, y1, y2 etc doesn't offer any advantage over tuples. Returning a ReturnValue instance with properties .y0 .y1 .y2 etc doesn't offer any advantage over tuples either. You need to start naming things if you want to get anywhere, and you can do that using tuples anyway:

```
def getImageData(filename):
  [snip]
  return size, (format, version, compression), (width,height)
size, type, dimensions = getImageData(x)
```

IMHO, the only good technique beyond tuples is to return real objects with proper methods and properties, like you get from `re.match()` or `open(file)`.

answered Dec 10 '08 at 2:22
too much php
38.6k ● 25 ● 98 ● 122

1 Question - is there any difference between `size, type, dimensions = getImageData(x)` and `(size, type, dimensions) = getImageData(x)`? I.e., does wrapping left-hand-side of a tupled assignment make any difference? – Reb.Cabin Dec 28 '14 at 17:15

1 @Reb.Cabin There is no difference. Tuple is identified by comma and the use of parentheses is just to group things together. For example `(1)` is a int while `(1,)` or `1,` is a tuple. – phil May 14 '15 at 7:04

I vote for the dictionary.

I find that if I make a function that returns anything more than 2-3 variables I'll fold them up in a dictionary. Otherwise I tend to forget the order and content of what I'm returning.

Also, introducing a 'special' structure makes your code more difficult to follow. (Someone else will have to search through the code to find out what it is)

If your concerned about type look up, use descriptive dictionary keys, for example, 'x-values list'.

```
def g(x):
  y0 = x + 1
  y1 = x * 3
  y2 = y0 ** y3
  return {'y0':y0, 'y1':y1 ,'y2':y2 }
```

answered Dec 10 '08 at 2:42
monkut
18.7k ● 8 ● 63 ● 104

after many years of programming, I tend toward what ever the structure of the data and function is required. Function first, you can always refactor as is necessary. – monkut Jun 3 '14 at 2:59

How would we get the values within the dictionary without calling the function multiple times? For example, if I want to use y1 and y3 in a different function? – HEATH3N Nov 2 '14 at 1:19

1 assign the results to a separate variable. `result = g(x); other_function(result)` – monkut Nov 4 '14 at 1:09

I prefer to use tuples whenever a tuple feels "natural"; coordinates are a typical example, where the separate objects can stand on their own, e.g. in one-axis only scaling calculations.

I use dictionaries as a return value only when the grouped objects aren't always the same. Think optional email headers.

For the rest of the cases, where the grouped objects have inherent meaning inside the group or a fully-fledged object with its own methods is needed, I use a class.

answered Dec 10 '08 at 2:40
tzot
43.2k ● 14 ● 78 ● 143

+1 on S.Lott's suggestion of a named container class.

For python 2.6 and up, a named tuple provides a useful way of easily creating these container classes, and the results are "lightweight and require no more memory than regular tuples".

answered Dec 10 '08 at 3:51
John Fouhy
20.4k ● 8 ● 44 ● 64

Thanks, I hadn't heard of named tuples in python yet. I suppose their greatest usefulness is in interfacing with existing code that returns tuples. – saffsd Dec 10 '08 at 4:38

Another option would be using generators:

```
>>> def f(x):
        y0 = x + 1
        yield y0
        yield x * 3
        yield y0 ** 4


>>> a, b, c = f(5)
>>> a
6
>>> b
15
>>> c
1296
```

Although IMHO tuples are usually best, except in cases where the values being returned are candidates for encapsulation in a class.

answered Feb 23 '14 at 15:26

sweeneyrod
4,341 ●3 ●14 ●42

This seems to be the cleanest solution, and has a clean syntax. Are there any downsides to this? If you don't use all the returns, are there 'unspent' yields waiting to hurt you? – Jiminion Dec 16 '15 at 14:08

This may be "clean", but it doesn't seem intuitive at all. How would someone who's never encountered this pattern before know that doing automatic tuple unpacking would trigger each `yield` ? – CoreDumpError Jan 6 at 21:38

```
>>> def func():
...     return [1,2,3]
...
>>> a,b,c = func()
>>> a
1
>>> b
2
>>> c
3
```

answered Jan 21 '15 at 20:57

WebQube
1,615 ●2 ●14 ●32

1 Notice return 1,2,3 gets the same as return [1,2,3] – edouard Nov 3 '15 at 16:52

I prefer

```
def g(x):
  y0 = x + 1
  y1 = x * 3
  y2 = y0 ** y3
  return {'y0':y0, 'y1':y1 ,'y2':y2 }
```

it seems everything else is just extra code to do the same thing.

answered Dec 10 '08 at 2:00

UnkwnTech
29.3k ●51 ●139 ●205

8 Tuples are easier to unpack: y0, y1, y2 = g() with a dict you have to do: result = g() y0, y1, y2 = result.get('y0'), result.get('y1'), result.get('y2') which is a little bit ugly. Each solution has its 'pluses' and its 'minuses'. – Oli Dec 10 '08 at 7:46

Generally, the "specialized structure" actually IS a sensible current state of an object, with its own methods.

```
class Some3SpaceThing(object):
  def __init__(self,x):
    self.g(x)
  def g(self,x):
    self.y0 = x + 1
    self.y1 = x * 3
    self.y2 = y0 ** y3

r = Some3SpaceThing( x )
r.y0
r.y1
r.y2
```

I like to find names for anonymous structures where possible. Meaningful names make things more clear.

edited Dec 10 '08 at 2:33
tzot
43.2k ● 14 ● 78 ● 143

answered Dec 10 '08 at 2:15
S.Lott
247k ● 50 ● 350 ● 623

Python's tuples, dicts, and objects offer the programmer a smooth tradeoff between formality and convenience for small data structures ("things"). For me, the choice of how to represent a thing is dictated mainly by how I'm going to use the structure. In C++, it's a common convention to use `struct` for data-only items and `class` for objects with methods, even though you can legally put methods on a `struct`; my habit is similar in Python, with `dict` and `tuple` in place of `struct`.

For coordinate sets, I'll use a `tuple` rather than a point `class` or a `dict` (and note that you can use a `tuple` as a dictionary key, so `dict`s make great sparse multidimensional arrays).

If I'm going to be iterating over a list of things, I prefer unpacking `tuple`s on the iteration:

```
for score,id,name in scoreAllTheThings():
    if score > goodScoreThreshold:
        print "%6.3f #%6d %s"%(score,id,name)
```

...as the object version is more cluttered to read:

```
for entry in scoreAllTheThings():
    if entry.score > goodScoreThreshold:
        print "%6.3f #%6d %s"%(entry.score,entry.id,entry.name)
```

...let alone the `dict`.

```
for entry in scoreAllTheThings():
    if entry['score'] > goodScoreThreshold:
        print "%6.3f #%6d %s"%(entry['score'],entry['id'],entry['name'])
```

If the thing is widely used, and you find yourself doing similar non-trivial operations on it in multiple places in the code, then it's usually worthwhile to make it a class object with appropriate methods.

Finally, if I'm going to be exchanging data with non-Python system components, I'll most often keep them in a `dict` because that's best suited to JSON serialization.

answered Aug 13 '13 at 20:18
Russell Borogove
10.2k ● 19 ● 31

In languages like Python, I would usually use a dictionary as it involves less overhead than creating a new class.

However, if I find myself constantly returning the same set of variables, then that probably involves a new class that I'll factor out.

answered Dec 10 '08 at 2:02
fluffels
2,103 ● 5 ● 23 ● 45
