Alternatives for returning multiple values from a Python function [closed]

Asked 14 years, 2 months ago   Modified 6 months ago   Viewed 1.4m times

1213

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}
```

(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:

```python
class ReturnValue:
    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:

```python
class ReturnValue(object):
    __slots__ = ["y0", "y1", "y2"]
    def __init__(self, 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 variable. Space is saved because `__dict__` is not created for each instance.

Option: Using a [dataclass](#) (Python 3.7+)

Using Python 3.7's new dataclasses, return a class with automatically added special methods, typing and other useful tools:

```python
@dataclass
class Returnvalue:
    y0: int
    y1: float
    y3: int

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

Option: Using a list

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

```python
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, whereas 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?

python   coding-style   return   return-value

Share Follow

edited Aug 10, 2022 at 18:56

divenex
13.9k   9   53   54

asked Dec 10, 2008 at 1:55

saffsd
23.5k   17   63   66

---

10   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, 2013 at 14:49

@hetepeperfan no need to change 3, and neither is defining y3 in global, you could also use a local name `y3`, that will do the same job too. – okie Oct 27, 2019 at 23:31

---

14 Answers

Sorted by:

Highest score (default) ⇕

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

689

```python
>>> 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
```

In recent versions of Python 3 (3.6+, I think), the new `typing` library got the `NamedTuple` class to make named tuples easier to create and more powerful. Inheriting from `typing.NamedTuple` lets you use

docstrings, default values, and type annotations.

Example (From the docs):

```python
class Employee(NamedTuple):  # inherit from typing.NamedTuple
    name: str
    id: int = 3  # default value

employee = Employee('Guido')
assert employee.id == 3
```

Share Follow

edited Nov 27, 2019 at 0:13

ShadowRanger
138k 12 175 258

answered Dec 10, 2008 at 16:36

A. Coady
52.9k 8 33 39

---

7 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, 2016 at 17:07

2 Best answer I think. One thing that I didn't realize immediately - you don't need to declare the namedtuple in the outer scope; your function itself can define the container and return it. – wom Dec 15, 2016 at 14:05

12 @wom: Don't do this. Python makes no effort to uniquify namedtuple definitions (each call creates a new one), creating the namedtuple class is relatively expensive in both CPU and memory, and all class definitions intrinsically involve cyclic references (so on CPython, you're waiting for a cyclic GC run for them to be released). It also makes it impossible to pickle the class (and therefore, impossible to use instances with multiprocessing in most cases). Each creation of the class on my 3.6.4 x64 consumes ~0.337 ms, and occupies just under 1 KB of memory, killing any instance savings. – ShadowRanger Apr 20, 2018 at 19:20 ✏️

6 I will note, Python 3.7 improved the speed of creating new namedtuple classes; the CPU costs drop by roughly a factor of 4x, but they're still roughly 1000x higher than the cost to create an instance, and the memory cost for each class remains high (I was wrong in my last comment about "under 1 KB" for the class, _source by itself is typically 1.5 KB; _source is removed in 3.7, so it's likely closer to the original claim of just under 1 KB per class creation). – ShadowRanger Apr 20, 2018 at 19:31 ✏️

2 @endolith because you can add values after you create it, meaning you can add the results to the retval Namespace as soon as you get them and don't have to wait until you can put them in a named tuple all at once. Can reduce clutter in bigger functions by a lot sometimes. – jaaq Oct 27, 2019 at 12:45

264

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:

```python
def get_image_data(filename):
    [snip]
    return size, (format, version, compression), (width,height)

size, type, dimensions = get_image_data(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)`.

Share Follow

27   Re "returning a dictionary with keys y0, y1, y2 etc doesn't offer any advantage over tuples": dictionary has the advantage in that you can add fields to the returned dictionary without breaking existing code. – ostrokach Aug 28, 2016 at 21:10

6   Re "returning a dictionary with keys y0, y1, y2 etc doesn't offer any advantage over tuples": it's also more readable and less error prone as you access the data based on its name rather than position. – Denis Dollfus Jan 16, 2020 at 13:09

239

A lot of the answers suggest you need to return a collection of some sort, like a dictionary or a list. You could leave off the extra syntax and just write out the return values, comma-separated. Note: this technically returns a tuple.

```
def f():
    return True, False
x, y = f()
print(x)
print(y)
```

gives:

```
True
False
```

Share Follow

27   You are still returning a collection. It is a tuple. I prefer parenthesis to make it more explicit. Try this: `type(f())` returns `<class 'tuple'>` . – Igor May 17, 2016 at 18:52 ✎

28   @Igor: There is no reason to make the `tuple` aspect explicit; it's not really important that you're returning a `tuple` , this is the idiom for returning multiple values period. Same reason you omit the parens with the swap idiom, `x, y = y, x` , multiple initialization `x, y = 0, 1` , etc.; sure, it makes `tuple` s under the hood, but there is no reason to make that explicit, since the `tuple` s aren't the point at all. The Python tutorial introduces multiple assignment long before it even touches on `tuple` s. – ShadowRanger Apr 20, 2018 at 19:39 ✎

@ShadowRanger any sequence of values separated by a comma in the right hand side of `=` is a tuple in Python with or without parenthesis around them. So there's actually no explicit or implicit here. a,b,c is as much a tuple as (a,b,c). There's also no making of tuple "under the hood" when you return such values, because it's just a plain simple tuple. The OP already mentioned tuples so there's actually no difference between what he mentioned and what this answer shows. None – Ken4scholars Jan 23, 2019 at 19:25

2   This is the literally the first option suggested in the question – endolith Oct 26, 2019 at 15:43

1   @endolith The two times the guy asks a question ("How do I return multiple values?" and "How do *you* return multiple values?") are answered by this answer. The text of the question has changed sometimes. And it's an opinion-based question. – Joe Hansen Feb 19, 2020 at 14:22

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'.

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

Share Follow

answered Dec 10, 2008 at 2:42

monkut
41.2k 24 122 149

---

6   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, 2014 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? – Matt Nov 2, 2014 at 1:19

4   assign the results to a separate variable. `result = g(x); other_function(result)` – monkut Nov 4, 2014 at 1:09

1   @monkut yes. This way also allows to pass result to several functions, which take different args from result, without having to specifically reference particular parts of the result every time. – Gnudiff Jan 7, 2018 at 22:45
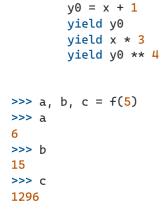

Another option would be using generators:

```python
>>> 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.

Share Follow

answered Feb 23, 2014 at 15:26

rlms
10.4k 8 43 60

I prefer:

38

```
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.

Share　Follow

edited Jun 20, 2019 at 2:00
Peter Mortensen
31k　21　105　125

answered Dec 10, 2008 at 2:00
UnkwnTech
86.4k　65　183　229

31

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, and the order is important. Note: if I can sort or shuffle the items without an adverse effect to the meaning of the group, then I probably shouldn't use a tuple.

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.

Share　Follow

edited May 30, 2018 at 16:53
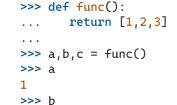
answered Dec 10, 2008 at 2:40
tzot
90.8k　29　140　201

29

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

```
2
>>> c
3
```

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

```python
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.

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:

```python
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:

```python
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.

Share Follow

answered Aug 13, 2013 at 20:18

Russell Borogove
18.2k　4　42　49

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

20

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".

Share Follow

edited Jun 20, 2019 at 2:03

Peter Mortensen
31k　21　105　125

answered Dec 10, 2008 at 3:51

John Fouhy
40.5k　19　62　77

"Best" is a partially subjective decision. Use tuples for small return sets in the general case where an immutable is acceptable. A tuple is always preferable to a list when mutability is not a requirement.

5

For more complex return values, or for the case where formality is valuable (i.e. high value code) a named tuple is better. For the most complex case an object is usually best. However, it's really the situation that matters. If it makes sense to return an object because that is what you naturally have at the end of the function (e.g. Factory pattern) then return the object.

As the wise man said:

> Premature optimization is the root of all evil (or at least most of it) in programming.

Share Follow

answered Aug 29, 2018 at 19:30

joel3000
1,189　10　22

---

2 Donald Knuth said it (in 1974). – Peter Mortensen Jun 20, 2019 at 2:07

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

4

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

Share Follow

answered Dec 10, 2008 at 2:02

I would use a dict to pass and return values from a function:

Use variable form as defined in [form](#).

```
form = {
    'level': 0,
    'points': 0,
    'game': {
        'name': ''
    }
}


def test(form):
    form['game']['name'] = 'My game!'
    form['level'] = 2

    return form

>>> print(test(form))
{u'game': {u'name': u'My game!'}, u'points': 0, u'level': 2}
```

This is the most efficient way for me and for processing unit.

You have to pass just one pointer in and return just one pointer out.

You do not have to change functions' (thousands of them) arguments whenever you make a change in your code.

Share Follow

edited Oct 30, 2017 at 14:06

answered Oct 30, 2017 at 14:01

Elis Byberi
1,410 1 11 20

---

1 Dicts are mutable. If you pass a dict to a function and that function edits the dict, changes will be reflected outside the scope of that function. Having the function return the dict at the end might imply that the function has no side-effects, therefore the value should not be returned, making it clear that `test` will directly modify the value. Compare this with `dict.update`, which does not return a value. – sleblanc Jan 28, 2019 at 7:01

@sleblanc "Having the function return the dict at the end might imply that the function has no side-effects". It does not imply that because ,as you said, dict is mutable. However, returning `form` does not hurt readability nor performance. In cases where you may need to reformat `form`, returning it [form] does make sure that the last `form` is returned because you will not keep track of form changes anywhere. – Elis Byberi Jan 28, 2019 at 14:03

🔥 **Highly active question**. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.