Saving an Object (Data persistence)

I've created an object like this:

```
company1.name = 'banana'
company1.value = 40
```

I would like to save this object. How can I do that?

python object save pickle

edited Jan 7 at 1:09 — martineau — 48.6k 6 68 115

asked Dec 25 '10 at 9:02 — Peterstone — 1,434 10 30 44

2 Answers

You could use the `pickle` module in the standard library. Here's an elementary application of it to your example:

```
import pickle

class Company(object):
    def __init__(self, name, value):
        self.name = name
        self.value = value

with open('company_data.pkl', 'wb') as output:
    company1 = Company('banana', 40)
    pickle.dump(company1, output, pickle.HIGHEST_PROTOCOL)

    company2 = Company('spam', 42)
    pickle.dump(company2, output, pickle.HIGHEST_PROTOCOL)

del company1
del company2

with open('company_data.pkl', 'rb') as input:
    company1 = pickle.load(input)
    print(company1.name)  # -> banana
    print(company1.value)  # -> 40

    company2 = pickle.load(input)
    print(company2.name) # -> spam
    print(company2.value)  # -> 42
```

You could also use a simple utility like the following which opens a file and writes a single object to it:

```
def save_object(obj, filename):
    with open(filename, 'wb') as output:
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

# sample usage
save_object(company1, 'company1.pkl')
```

Update:

Since this is such a popular answer, I'd like touch on a few slightly advanced usage topics.

First, it's almost always preferable to actually use the `cPickle` module rather than `pickle` because the former is written in C and is much faster. There are some subtle differences between them, but in most situations they're equivalent and the C version will provide greatly superior performance. Switching to it couldn't be easier, just change the `import` statement to this:

```
import cPickle as pickle
```

(**Note:** In Python 3, `cPickle` was renamed `_pickle`, but doing this is no longer necessary because the `pickle` module now does it automatically — see question *What difference between pickle and _pickle in python 3?*).

Secondly, instead of writing `pickle.HIGHEST_PROTOCOL` in every call (assuming that's what you want, and you usually do), you can instead just use the literal `-1`. So, instead of writing:

```
pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)
```

You can just write:

```
pickle.dump(obj, output, -1)
```

Which is quite a bit shorter.

An even better way where you only have specify the protocol once is to create a `Pickler` object and use it to do multiple pickle operations:

```
pickler = pickle.Pickler(output, -1)
pickler.dump(obj1)
pickler.dump(obj2)
   etc...
```

Thirdly, while a pickle file *can* contain any number of pickled objects, as shown in the above samples, when there's an unknown number of them, it's often easier to store them all in some sort of variably-sized container, like a `list`, `tuple`, or `dict` and write them all to the file in a single call:

```
tech_companies = [
    Company('Apple', 114.18), Company('Google', 908.60), Company('Microsoft',
69.18)
]
save_object(tech_companies, 'tech_companies.pkl')
```

and restore the list and everything in it later with:

```
with open('tech_companies.pkl', 'rb') as input:
    tech_companies = pickle.load(input)
```

The major advantage is you don't need to know how many object instances are saved in order to load them back later (although doing so without that information *is* possible, it requires some specialized code). See the answers to the related question *Saving and loading multiple objects in pickle file?* for details.

edited yesterday

answered Dec 25 '10 at 9:35

martineau
48.6k 6 68 115

I think it's a pretty strong assumption to assume that the object is a `class`. What if it's not a `class`? There's also the assumption that the object was not defined in the interpreter. What if it was defined in the interpreter? Also, what if the attributes were added dynamically? When some python objects have attributes added to their `__dict__` after creation, `pickle` doesn't respect the addition of those attributes (i.e. it 'forgets' they were added -- because `pickle` serializes by reference to the object definition).

In all these cases, `pickle` and `cPickle` can fail you horribly.

If you are looking to save an `object` (arbitrarily created), where you have attributes (either added in the object definition, or afterward)… your best bet is to use `dill`, which can serialize almost anything in python.

We start with a class…

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> class Company:
...     pass
...
>>> company1 = Company()
>>> company1.name = 'banana'
>>> company1.value = 40
>>> with open('company.pkl', 'wb') as f:
...     pickle.dump(company1, f, pickle.HIGHEST_PROTOCOL)
...
>>>
```

Now shut down, and restart...

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> with open('company.pkl', 'rb') as f:
...     company1 = pickle.load(f)
...
Traceback (most recent call last):
```

```
  File "<stdin>", line 2, in <module>
  File
"/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/pickle.py
 line 1378, in load
    return Unpickler(file).load()
  File
"/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/pickle.py
 line 858, in load
dispatch[key](self)
  File
"/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/pickle.py
 line 1090, in load_global
    klass = self.find_class(module, name)
  File
"/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/pickle.py
 line 1126, in find_class
    klass = getattr(mod, name)
AttributeError: 'module' object has no attribute 'Company'
>>>
```

Oops… `pickle` can't handle it. Let's try `dill`. We'll throw in another object type (a `lambda`) for good measure.

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import dill
>>> class Company:
...     pass
...
>>> company1 = Company()
>>> company1.name = 'banana'
>>> company1.value = 40
>>>
>>> company2 = lambda x:x
>>> company2.name = 'rhubarb'
>>> company2.value = 42
>>>
>>> with open('company_dill.pkl', 'wb') as f:
...     dill.dump(company1, f)
...     dill.dump(company2, f)
...
>>>
```

And now read the file.

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import dill
>>> with open('company_dill.pkl', 'rb') as f:
...     company1 = dill.load(f)
...     company2 = dill.load(f)
...
>>> company1
<__main__.Company instance at 0x107909128>
>>> company1.name
'banana'
>>> company1.value
40
>>> company2.name
'rhubarb'
>>> company2.value
42
>>>
```

It works. The reason `pickle` fails, and `dill` doesn't, is that `dill` treats `__main__` like a module (for the most part), and also can pickle class definitions instead of pickling by reference (like `pickle` does). The reason `dill` can pickle a `lambda` is that it gives it a name… then pickling magic can happen.

Actually, there's an easier way to save all these objects, especially if you have a lot of objects you've created. Just dump the whole python session, and come back to it later.

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import dill
>>> class Company:
...     pass
...
>>> company1 = Company()
>>> company1.name = 'banana'
>>> company1.value = 40
>>>
>>> company2 = lambda x:x
>>> company2.name = 'rhubarb'
>>> company2.value = 42
>>>
>>> dill.dump_session('dill.pkl')
>>>
```

Now shut down your computer, go enjoy an espresso or whatever, and come back later...

```
Python 2.7.8 (default, Jul 13 2014, 02:29:54)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import dill
>>> dill.load_session('dill.pkl')
>>> company1.name
'banana'
>>> company1.value
40
>>> company2.name
'rhubarb'
>>> company2.value
42
>>> company2
<function <lambda> at 0x1065f2938>
```

The only major drawback is that `dill` is not part of the python standard library. So if you can't install a python package on your server, then you can't use it.

However, if you are able to install python packages on your system, you can get the latest `dill` with `git+https://github.com/uqfoundation/dill.git@master#egg=dill`. And you can get the latest released version with `pip install dill`.

edited May 13 '16 at 0:54

hobs
7,099 4 47 56

answered Aug 4 '14 at 12:49

Mike McKerns
11.1k 1 45 60