Pickle data serialisation

All Versions

Examples

Using Pickle to serialize and deserialize an object

The pickle module implements an algorithm for turning an arbitrary Python object into a series of bytes. This process is also called **serializing** the object. The byte stream representing the object can then be transmitted or stored, and later reconstructed to create a new object with the same characteristics.

For the simplest code, we use the dump() and load() functions.

To serialize the object

```
import pickle

# An arbitrary collection of objects supported by pickle.
data = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ("character string", b"byte string"),
    'c': {None, True, False}
}

with open('data.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
```

To deserialize the object

```
import pickle

with open('data.pickle', 'rb') as f:
    # The protocol version used is detected automatically, so we do not
    # have to specify it.
    data = pickle.load(f)
```

Using pickle and byte objects

It is also possible to serialize into and deserialize out of byte objects, using the dumps and loads function, which are equivalent to dump and load .

```
serialized_data = pickle.dumps(data, pickle.HIGHEST_PROTOCOL)
# type(serialized_data) is bytes

deserialized_data = pickle.loads(serialized_data)
# deserialized_data == data
```

Customize Pickled Data

Some data cannot be pickled. Other data should not be pickled for other reasons.

What will be pickled can be defined in __getstate__ method. This method must return something that is picklable.

On the oposite side is __setstate__ : it will receive what __getstate__ created and has to initialize the object.

```
class A(object):
    def __init__(self, important_data):
        self.important_data = important_data

        # Add data which cannot be pickled:
        self.func = lambda: 7

        # Add data which should never be pickled, because it expires quickly:
        self.is_up_to_date = False

    def __getstate__(self):
        return [self.important_data] # only this is needed

    def __setstate__(self, state):
        self.important_data = state[0]

        self.func = lambda: 7  # just some hard-coded unpicklable function

        self.is_up_to_date = False  # even if it was before pickling
```

Now, this can be done:

```
>>> a1 = A('very important')
>>>
>>> s = pickle.dumps(a1)  # calls a1.__getstate__()
>>>
>>> a2 = pickle.loads(s)  # calls a1.__setstate__(['very important'])
>>> a2
<__main__.A object at 0x0000000002742470>
>>> a2.important_data
'very important'
>>> a2.func()
7
```

The implementation here pikles a list with one value: `[self.important_data]` . That was just an example, `__getstate__` could have returned anything that is picklable, as long as `__setstate__` knows how to do the oppoisite. A good alternative is a dictionary of all values: `{'important_data': self.important_data}` .

Constructor is not called! Note that in the previous example instance `a2` was created in `pickle.loads` without ever calling `A.__init__` , so `A.__setstate__` had to initialize everything that `__init__` would have initialized if it were called.

Syntax

```
pickle.dump(object,file,protocol) #To serialize an object
pickle.load(file) #To de-serialize an object
pickle.dumps(object, protocol) # To serialize an object to bytes
pickle.loads(buffer) # To de-serialzie an object from bytes
```

Parameters

Parameter	Details
object	The object which is to be stored
file	The open file which will contain the object
protocol	The protocol used for pickling the object (optional parameter)
buffer	A bytes object that contains a serialized object

Remarks

Pickleable types

The following objects are picklable.

- `None` , `True` , and `False`
- numbers (of all types)
- strings (of all types)
- `tuple` s, `list` s, `set` s, and `dict` s containing only picklable objects
- functions defined at the top level of a module
- built-in functions
- classes that are defined at the top level of a module
  - instances of such classes whose `__dict__` or the result of calling `__getstate__()` is picklable (see the official docs for details).

Based on the official Python documentation .

`pickle` and security

The pickle module is **not secure** . It should not be used when receiving the serialized data from an untrusted party, such as over the Internet.