Pickles are for Delis, Not Software

Alex Gaynor

About me

- Django, PyPy, OpenStack, CPython, etc.
- Director of the Python Software Foundation
- Software engineer at Rackspace

These are a few of my favorite things…


HEINZ
TOMATO
2ND AVE DELI
RESTAURANT & CATERERS

```
p = password.ToCharArray
getLength(0);
] bt = new byte[len];
=0; i<len ; i++)
bt[i] = (byte) cp[
umKey = new byte [8
= new byte [
sha = new Sh
st );
()
```

These are a few of my least favorite things…

```
>>> import pickle
```

What is pickle?

- Serialization library
- Take a random python object, turn it into bytes
- Take some bytes, turn them back into an object later

Use cases

- Sending things between two Python processes which are both running (e.g. multiprocessing)

- Storing random Python objects in your database (e.g. memcached)

So how’s it work?

```
>>> pickle.dumps([1, "a", None])
"(lp0\nI1\naS'a'\np1\naNa."
```

```
>>> pickle.dumps([1, "a", None])
"(lp0\nI1\nas
```

Random nonsense!

```
def dump(self, obj):
    # Check the type dispatch table
    t = type(obj)
    f = self.dispatch.get(t)
    if f:
        f(self, obj)
        return
    ...
```

```
def dump_list(self, obj):
        self.write(EMPTY_LIST)
        for x in obj:
            self.dump(x)
        self.write(APPEND)
```

```
def dump_int(self, obj):
    self.write(INT + repr(obj) + '\n')
```

```
def dump_none(self, obj):
    self.write(NONE)
```

```
>>> import pickletools
>>> pickletools.dis("(lp0\nI1\naS'a'\np1\naNa.")
    1: l          LIST       (MARK at 0)
    5: I      INT            1
    8: a      APPEND
    9: S      STRING     'a'
   17: a      APPEND
   18: N      NONE
   19: a      APPEND
   20: .      STOP
```

So what about your classes?

```
def dump(self, obj):
    # Check the type dispatch table
    t = type(obj)
    f = self.dispatch.get(t)
    if f:
        f(self, obj)
        return
    ...
```

```
def dump(self, obj):
    # ...
    func, args = obj.__reduce__()
    if reduce:
        rv = reduce()
    self.save(func)
    self.save(args)
    self.write(REDUCE)
```

```
>>> pickletools.dis(pickle.dumps(object()))
    0: c    GLOBAL     'copy_reg _reconstructor'
   29: c        GLOBAL     '__builtin__ object'
   55: N        NONE
   56: t        TUPLE
   60: R    REDUCE
   64: .    STOP
```

```
>>> class X(object):
...     def __init__(self):
...         self.my_cool_attr = 3
...
>>> x = X()
```

```
>>> pickletools.dis(pickle.dumps(x))
    0: c    GLOBAL     'copy_reg _reconstructor'
   29: c        GLOBAL     '__main__ X'
   44: c        GLOBAL     '__builtin__ object'
   67: N        NONE
   68: t        TUPLE
   72: R    REDUCE
   77: d        DICT
   81: S    STRING     'my_cool_attr'
  100: I    INT        3
  103: s    SETITEM
  104: b    BUILD
  105: .    STOP
```

What if I want to have custom pickle logic?

```
>>> class FunkyPickle(object):
...     def __reduce__(self):
...         return (str, ('abc',),)
...
...
```

```
>>> pickle.loads(pickle.dumps(FunkyPickle()))
'abc'
```

Unpickling

Stack machines

```
>>> [1, 'a', None]
[1, 'a', None]
```

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

[ ]

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

[ ]
1

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

```
[1]
```

LIST

INT 1

APPEND

STRING

```
[1]
```

```
'a'
```

APPEND

None

APPEND

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

```
[1, 'a']
```

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND


```
[1, 'a']
```

```
None
```

LIST

INT 1

APPEND

STRING

APPEND

None

APPEND

```
[1, 'a', None]
```

STOP

```
[1, 'a', None]
```

```
def loads(self):
    try:
        while True:
            key = self.read(1)
            self.dispatch[key](self)
    except _Stop as exc:
        return exc.value
```

```
def load_empty_list(self):
    self.stack.append([])
dispatch[EMPTY_LIST] = load_empty_list

def load_append(self):
    stack = self.stack
    value = stack.pop()
    list = stack[-1]
    list.append(value)
dispatch[APPEND] = load_append
```

```
def load_reduce(self):
    stack = self.stack
    args = stack.pop()
    func = stack[-1]
    value = func(*args)
    stack[-1] = value
dispatch[REDUCE] = load_reduce
```

Security

```
>>> class WhatEvenIsSecurity(object):
...  def __reduce__(self):
...   return (os.listdir, ('.',),)
...
>>> p = pickle.dumps(WhatEvenIsSecurity())
>>> pickle.loads(p)
['Applications', 'Desktop', 'Documents', 'Downloads', ...]
```

You cannot safely unpickle untrusted data.

https://blog.nelhage.com/2011/03/exploiting-pickle/

Do you, Programmer, take this Object to be part of the persistent state of your application, to have and to hold, through maintenance and iterations, for past and future versions, as long as the application shall live?

- Erm, can I get back to you on that?

Let me tell you a story…

```
>>> class X(object):
...  pass
...
>>> x = X()
>>> p = pickle.dumps(x)
>>> del X
>>> pickle.loads(p)
Traceback (most recent call last):
...
AttributeError: 'module' object has no attribute 'X'
```

```
>>> class X(object):
...   def __init__(self):
...     self.x = 4
...   def f(self):
...     return self.x
...
>>> p = pickle.dumps(X())
>>> pickle.loads(p)
<__main__.X object at 0x107f80d90>
```

```
>>> class X(object):
...  def __init__(self):
...   self.y = 4
...  def f(self):
...   return self.y
...
>>> x = pickle.loads(p)
>>> x.f()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in f
AttributeError: 'X' object has no attribute 'y'
```

Alternatives

(Also known as, write some functions)

```
class Table(object):
    def __init__(self, size):
        self.size = size

    def dump(self):
        return json.dumps({
            "version": 1,
            "size": self.size
        })
```

```
@classmethod
def load(cls, data):
    assert data["version"] == 1
    return cls(data["size"])
```

```
class Table(object):
    def __init__(self, height, width):
        self.heigh = heigh
        self.width = width

    def dump(self):
        return json.dumps({
            "version": 2,
            "height": self.height,
            "width": self.width,
        })
```

```
@classmethod
def load(cls, data):
    if data["version"] == 1:
        h = w = sqrt(data["size"])
        return cls(h, w)
    elif data["version"] == 2:
        return cls(data["height"], data["width"])
    else:
        raise ValueError
```

- More testable
- Simpler
- More auditable

- msgpack
- JSON

Pickle: Unsafe at any speed

Thanks!

Questions? Answers?