



science + computing

| A Bull Group Company

A decorative banner at the top of the slide. It features a collage of images: on the left, a close-up of red network cables plugged into a switch with 'P5 P15' labels; in the center, a woman with dark hair, wearing a pink top, smiling; and on the right, a blurred background of a modern office or lab setting. The banner is overlaid with a grid of light blue and white squares.

Advanced Pickling with Stackless Python and sPickle

Anselm Kruis | EuroPython 2011

science + computing ag

IT-Services for Complex Computing Environments

Tübingen | Munich | Berlin | Düsseldorf

Who and Why

Who

Name: Anselm Kruis

Profession: Senior Architect at science + computing

Location: Munich

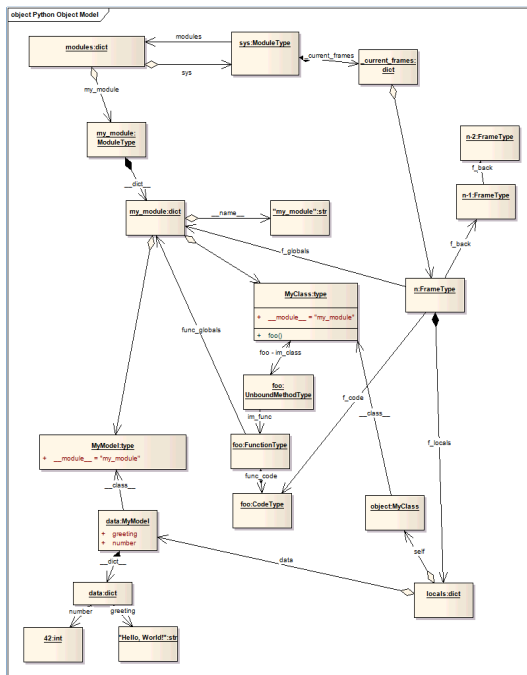
Why

- Python is fun, let's do some cool stuff
- How to migrate a running program from one computer to another?
- Is Pickling the way to go?
- Created sPickle

What is Pickling

Wikipedia:

In the computer programming language Python, pickle is the standard mechanism for object serialization; pickling is the common term among Python programmers for serialization (unpickling for deserialization).



pickle

'\x01\x02\x30....'

unpickle

How does pickling work?

- The Pickler writes a program in the Pickle language, a tiny but powerful programming language.
- The Unpickler is an interpreter for the Pickle language.

It creates a single complex object from primitive types and collections, imported objects, external objects and from the execution of already unpickled functions or methods. See module pickletools source for documentation.

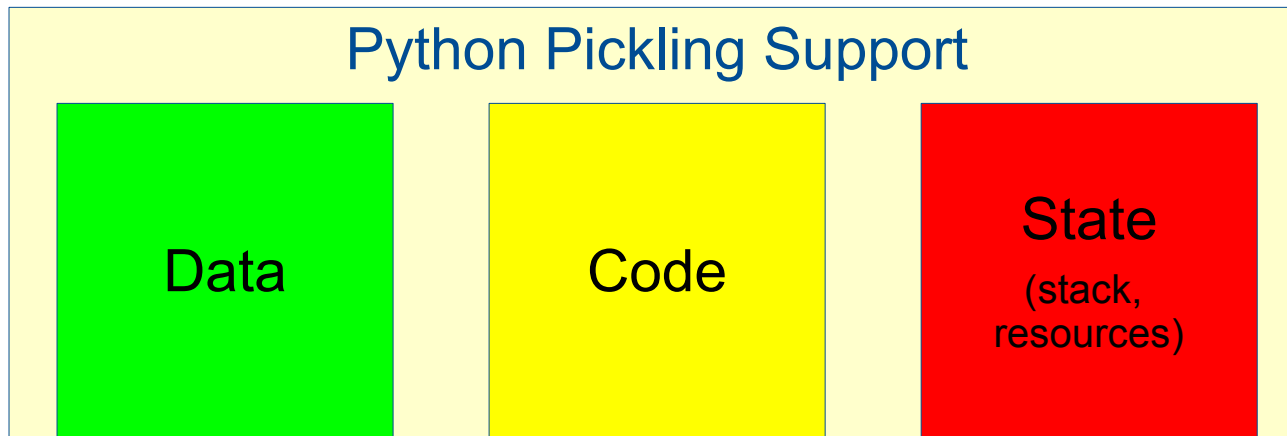
Unpickling is insecure. Do not unpickle untrusted data.

- To support new object types it is usually sufficient to improve the Pickler only.
- Both Stackless Python and the sPickle module extend the Pickler from the Python library.

What can be pickled by plain CPython?

Answer: Read the Python documentation (11.1.4):

- By value: data (strings, numbers, ... and collections of picklable objects)
- By reference: code (function, classes, ...) if they are importable
- Objects, which implement the pickle protocol.

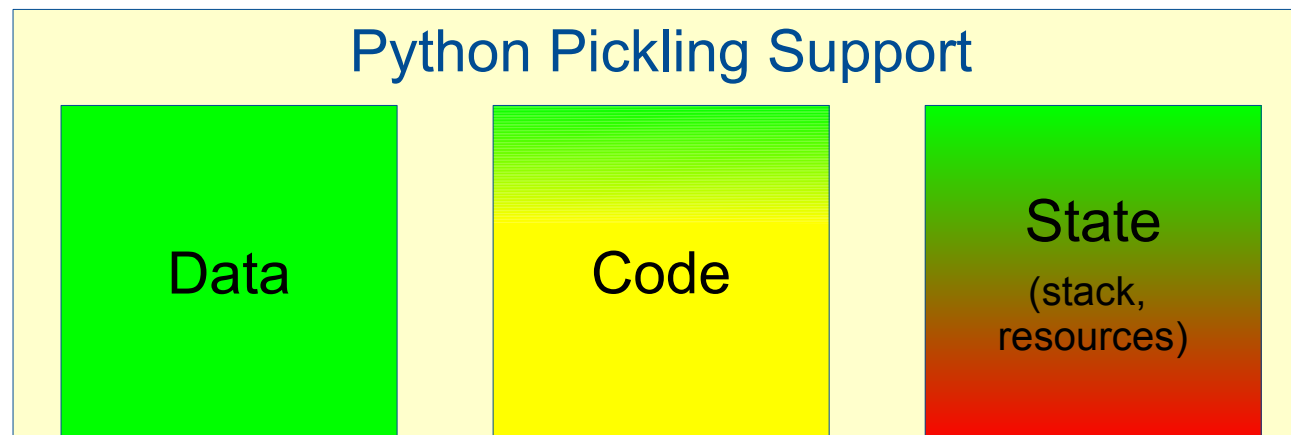


- by value
- by reference
- not at all

That is quite limited!

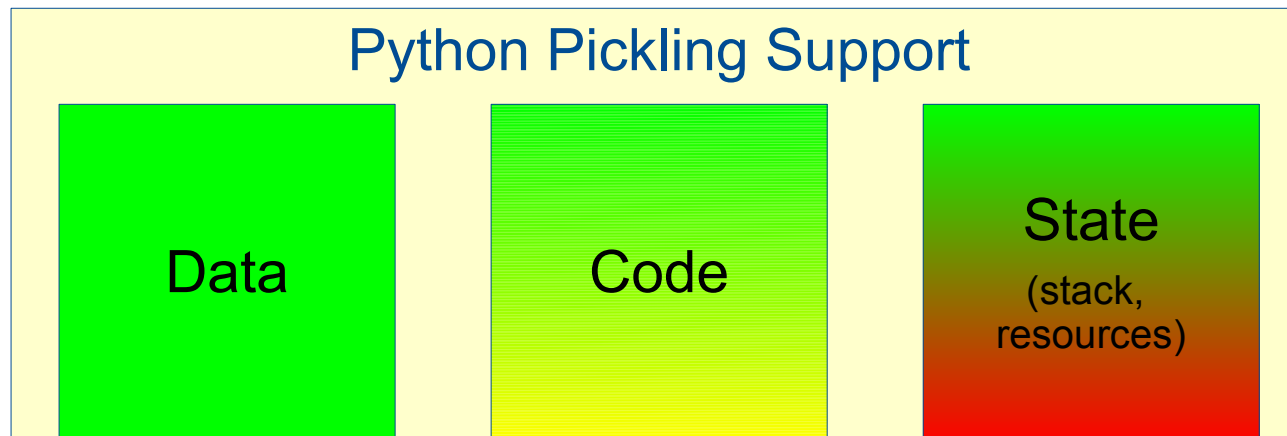
Stackless Python

- Stackless Python is a variant of CPython
- Python frame stack is independent from the C-Stack used by the CPU
- Tasklets (a kind of coroutine) are used to manage python stacks.
 - You can create many tasklets, each with its own frame-stack
 - You can switch control between tasklets
- Therefore the Stackless Python developers were able to implement pickling and unpickling for tasklets (and some other types)




Package sPickle

- It tries to fill the gap in the “Code” area.
- It provides a Pickler class, that is intended to replace the pickle.Pickler class. And a few utility classes.



- For unpickling we use the standard implementation from cPickle or pickle.

Package sPickle

- Name
 - a kind of super or stackless or smart pickle module
- Content
 - class Pickler
 - subclass of pickle.Pickler
 - mostly a plug in replacement of pickle.Pickler
 - class SPickleTools:
 - convenience functions for pickling and unpickling
 - functions, that didn't fit elsewhere
- Availability
 - <http://pypi.python.org/pypi/sPickle> Apache 2 License
 - Documentation <http://packages.python.org/sPickle>
 - Currently for Stackless Python 2.7 only 

Applications of Pickling

- Most important: Saving and restoring complex data.
 - Already possible with standard CPython
 - Mainstream, no longer advanced
- More interesting examples
 - Checkpointing of a program
(A kind of asynchronous migration of a program)
 - Remote Procedure Calls
(A kind of synchronous migration of a program)

Example 1: Checkpointing

- Definition (Wikipedia):

Checkpointing is a technique for inserting fault tolerance into computing systems. It basically consists of storing a snapshot of the current application state, and later on, use it for restarting the execution in case of failure.

- Implementation plan: pickle state, data objects and code

- Serialise nearly every object, including the python frame stack and some modules into a file.
- Restore everything from this file later on.
 - Python source code is no longer required
 - Can be on a different operating system
- Source: example1, included in the sPickle source archive

Checkpointing: Startup Code

```
import checkpointing

def long_running_function_with_checkpointing(checkpointSupport, *args, **keywords):
    print "At program start"
    ...
    while not isDone: # main loop
        ...
        isCmdResult, result = checkpointSupport.forkAndCheckpoint()
        if isCmdResult:
            # result is the pickle
            f = open(checkpointFile, "wb")
            f.write(result)
            f.close()
        else:
            # after restart
            ...

def main(argv):
    checkpointFile = "example1.pickle"
    ...
    from sPickle import SPickleTools
    # always serialize __main__, because the main used during a resume
    # operation is most likely a different module loaded from a different file
    pt = SPickleTools(serializeableModules=['__main__'])
    return checkpointing.runCheckpointable(pt.dumps,
                                           long_running_function_with_checkpointing,
                                           checkpointFile = checkpointFile,
                                           *argv)
```

Checkpointing: Module checkpointing.py

```
class _CheckpointSupport(object):
    def _taskletRun(self, trace, callable, args, keywords):
        raise sPickle.StacklessTaskletReturnValueException(
            callable(self, *args, **keywords))

    def _loop(self, tasklet, pickler):
        try:
            while True:
                tasklet.run()
                ...
                pickle = pickler((self, tasklet))
                ...
                tasklet.tempval = (True, sys.gettrace(), pickle)
        except sPickle.StacklessTaskletReturnValueException, e:
            return e.value

    def forkAndCheckpoint(self, cmd=CMD_CHECKPOINT):
        ...
        isCmdResult, trace, result = stackless.schedule(cmd)
        ...
        return (isCmdResult, result)

def runCheckpointable(pickler, callable, *args, **keywords):
    checkpointSupport = _CheckpointSupport()
    tasklet = stackless.tasklet(checkpointSupport._taskletRun)
    tasklet.setup(sys.gettrace(), callable, args, keywords)
    tasklet.tempval = None
    return checkpointSupport._loop(tasklet, pickler)
```

Checkpointing: Resume Code

```
import checkpointing

def main(argv):
    checkpointFile = "example1.pickle"
    ...
    # Resume the execution of the checkpoint
    # Note: This resume code does not define any functional logic.
    #       You can also use the checkpointing module to resume example
    return checkpointing.resumeCheckpoint(open(checkpointFile, "rb").read(), *argv)
```

From checkpointing.py

```
def resumeCheckpoint(checkpoint, *args, **keywords):
    pt = sPickle.SPickleTools()
    checkpointSupport, tasklet = pt.loads(checkpoint)
    tasklet.tempval = (False, sys.gettrace(), (args, keywords))
    return checkpointSupport._loop(tasklet, pt.dumps)
```

Demo

Example 2: Remote compute slave

We want to perform an operation on a remote compute slave

- On the slave, we have:
 - ssh access
 - Stackless Python and the RPyC package.

RPyC stands for Remote Python Calls, is available via PyPI and provides symmetric remote procedure calls.

- sPickle simplifies RPyC remote procedure calls
 - Creates the remote function and referenced objects
 - Transparently pickles function result
 - Handles resources: files, sockets and similar objects
 - Simple application

```
def local_func(...)  
    ...  
pt = sPickle.PickleTools()  
remote_func = pt.remotemethod(connection, local_func)  
result = remote_func(...)
```

Example 2: Algorithm

```
Result = collections.namedtuple("Result", "value error exception")
```

```
class ComputeTheAnswer(object):
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2
        self.result = self.error = self.exception = None

    def compute(self):
        try:
            self.result = self.param1 + self.param2
            if int(self.result) != 42:
                self.error = "Result is inconsistent with "+
                    "previous calculations!"
        except Exception, e:
            self.error = "Hey, you asked the wrong question. "+
                "Try again with different parameters!"
            self.exception = e

    def getResult(self):
        return Result(self.result, self.error, self.exception)
```


Example 2: Remote Procedure Call

```
# the directory sPickle/examples is not in sys.path, therefore
# it is not possible to import modules from this directory.
# Therefore ask the pickler to serialise those modules.
pt = sPickle.SPickleTools(serializeableModules=["sPickle/examples"])

remoteLogger = logging.getLogger("remoteLogger")
def function(param1, param2):
    remoteLogger.info("Starting function with parameters: %r, %r",
                     param1, param2)
    algorithm = ComputeTheAnswer(param1, param2)
    remoteLogger.info("Computing ...")
    algorithm.compute()
    remoteLogger.info("Computing is done.")
    return algorithm.getResult()

remote_function = pt.remotemethod(connection, function)

# Lets perform a few computations
r = remote_function(22, 20)
print "Result: ", r
r = remote_function("4", "1")
print "Result: ", r
r = remote_function("42", None)
print "Result: ", r
```

Example 2

Demo

The implementation of sPickle

The Problem with the `__dict__`

- A module object has the following attributes:
 - `__dict__`: type: dict, read-only
 - `__dict__.keys()`
- Usually there are references to both, the module object and its `__dict__`

```
import mod # mod is now a module object
from mod import function # a function
```

Here: `function.im_globals` is `mod.__dict__`

- If we recreate an object structure, we must create the module objects first, because we can't set the `__dict__` attribute.
- Problem: If we inspect a dictionary, it might be the `__dict__` of a module.
 - How to decide?

How to find the module for a given dictionary?

- Not reliable:
 - Use the content of the dictionary and apply heuristics.
 - Usually we are able to guess the right module name and locate the module object via `sys.modules`
 - Complicated, very unreliable
 - Test all modules in `sys.modules`
 - still unreliable, if we use `reload()`
- OK: Use backtracking
 - While pickling objects, keep a reference to every dictionary and if we later encounter a module for dictionary, back up and pickle the module first.
 - Implemented in method `dict_checkpoint` using Exceptions
 - This implementation fits into the existing code, but the performance is bad
 - A better implementation needs a two pass Pickler.

Module Life Cycle or How do we unpickle the module “__main__”?

Obviously we have a name collision

Module name collisions provide three problems

- `sys.modules` can hold only one module per name
- During unpickling the entry in `sys.modules` might be the wrong one.
- How to keep a module alive after unpickling, if we can't store a reference in `sys.modules`?

Remember, shutdown of a module clears the dictionary of the module.

My solution

- in `sys.modules` replace the original module by the new module during unpickling.
- In case of a collision, restore the old state of `sys.modules` after unpickling and store a reference to the new module in `sys.sPicklePreservedModules`

Pickling Classes / Types

Problems

- Many types are built in / defined in native code
 - sPickle knows about a lot of them, but not all
 - the types module is incomplete
- Using metaclasses, you can build types, that are hard to inspect.

sPickle

- currently looks at the `__dict__` attribute of a class/type.
- No special code for metaclasses
- Works for me, but not perfect.
- Test cases and patches are welcome

Pickling Classes / Types II

How should the Pickler encode the creation of a class?

```
members={ ... # add all members __init__, ...  
}  
cls = type(name, bases, members)
```

or

```
cls = type(name, bases, {})  
for k,v in members.items():  
    setattr(cls, k, v)
```

For most attributes, either method is fine. But some attributes are processed by the metaclass:

- Do not set at all: `__dict__`, `__class__`, members of type `DictProxyType`, `GetSetDescriptorType`, `MemberDescriptorType`
- Set in the constructor: `__slots__`, `__doc__`, `__module__`
- Set via `setattr()`: all other

sPickle contains special code for:

- all types from the module types
- WRAPPER_DESCRIPTOR_TYPE
- METHOD_DESCRIPTOR_TYPE
- METHOD_WRAPPER_TYPE
- LISTITERATOR_TYPE
- TUPLEITERATOR_TYPE
- RANGEITERATOR_TYPE
- SETITERATOR_TYPE
- sys.stdout
- sys.stderr
- sys.stdin
- sys.__stdout__
- sys.__stderr__
- sys.__stdin__

Special Types `isinstance(obj, ...)`

sPickle contains special code for objects of type:

- `type(object.__new__)` and `__name__` in (`'__new__'`, `'__subclasshook__'`)
- `types.BuiltinMethodType`
- `thread.LockType`
- `types.FileType`
- `socket.SocketType`
- `SOCKET_PAIR_TYPE`
- `WRAPPER_DESCRIPTOR_TYPE`
- `METHOD_DESCRIPTOR_TYPE`
- `staticmethod`
- `classmethod`
- `property`
- `operator.itemgetter`
- `operator.attrgetter`

- Handling of trace functions

If a frame has a trace function (`f_trace` attribute of a frame), reconstruct this function using `sys.gettrace`

- Special functions from `sys`

The `sys` module contains two entries for some functions:

- `excepthook`, `__excepthook__`
- `displayhook`, `__displayhook__`

Test both versions, if looking for a function definition

Further Development of sPickle

- Add public GIT repository!!!
- We use sPickle module in a commercial product
- We plan to provide bug fixes
- Sorry, no plans for a Python 3 port
- sPickle is licensed under the Apache License, because this license is suitable for contributions to Python.
- Any chance to get some parts into Stackless or plain CPython?

Conclusion

- It is indeed possible to extend the Pickler
- The Python standard library does not care very much about pickling
- sPickle
 - is still experimental
 - many special cases are rely on undocumented implementation details of Python
 - currently only for Stackless Python 2.7
- Most features of sPickle could be integrated into standard C-Python



science + computing

| A Bull Group Company



Many thanks for your kind attention.

Anselm Kruis

science + computing ag

www.science-computing.de

Telefon +49-7071-9457-0

info@science-computing.de