Debugging Pickle Files

What does the pickle protocol look like? Let's jump out of the Python Shell for a moment and take a look at that `entry.pickle` file we created. To the naked eye, it's mostly gibberish.

```
you@localhost:~/diveintopython3/examples$ ls -l entry.pickle
-rw-r--r-- 1 you  you  358 Aug  3 13:34 entry.pickle
you@localhost:~/diveintopython3/examples$ cat entry.pickle
comments_linkqNXtagsqXdiveintopythonqXdocbookqXhtmlq?qX publishedq?
XlinkXJhttp://diveintomark.org/archives/2009/03/27/dive-into-history-2009-edition
q   Xpublished_dateq
ctime
struct_time
?qRqXtitleqXDive into history, 2009 editionqu.
```

That wasn't terribly helpful. You can see the strings, but other datatypes end up as unprintable (or at least unreadable) characters. Fields are not obviously delimited by tabs or spaces. This is not a format you would want to debug by yourself.

```
shell = 1
print (shell)
#1

import pickletools
with open('entry.pickle', 'rb') as f:
     pickletools.dis(f)
#    0: \x80 PROTO      3
#    2: }    EMPTY_DICT
#    3: q    BINPUT     0
#    5: (    MARK
#    6: X        BINUNICODE 'published_date'
#   25: q        BINPUT     1
#   27: c        GLOBAL     'time struct_time'
#   45: q        BINPUT     2
#   47: (        MARK
#   48: M            BININT2    2009
#   51: K            BININT1    3
#   53: K            BININT1    27
#   55: K            BININT1    22
#   57: K            BININT1    20
#   59: K            BININT1    42
```

```
#   61: K            BININT1    4
#   63: K            BININT1    86
#   65: J            BININT     -1

#   70: t            TUPLE      (MARK at 47)
#   71: q        BINPUT     3
#   73: }        EMPTY_DICT
#   74: q        BINPUT     4
#   76: \x86     TUPLE2
#   77: q        BINPUT     5
#   79: R        REDUCE
#   80: q        BINPUT     6
#   82: X        BINUNICODE 'comments_link'
#  100: q        BINPUT     7
#  102: N        NONE
#  103: X        BINUNICODE 'internal_id'
#  119: q        BINPUT     8
#  121: C        SHORT_BINBYTES 'ÞÕ´ø'
#  127: q        BINPUT     9
#  129: X        BINUNICODE 'tags'
#  138: q        BINPUT     10
#  140: X        BINUNICODE 'diveintopython'
#  159: q        BINPUT     11
#  161: X        BINUNICODE 'docbook'
#  173: q        BINPUT     12
#  175: X        BINUNICODE 'html'
#  184: q        BINPUT     13
#  186: \x87     TUPLE3
#  187: q        BINPUT     14
#  189: X        BINUNICODE 'title'
#  199: q        BINPUT     15
#  201: X        BINUNICODE 'Dive into history, 2009 edition'
#  237: q        BINPUT     16
#  239: X        BINUNICODE 'article_link'
#  256: q        BINPUT     17
#  258: X        BINUNICODE 'http://diveintomark.org/archives/2009/03/27/dive-into-history-20
#  337: q        BINPUT     18
#  339: X        BINUNICODE 'published'
#  353: q        BINPUT     19
#  355: \x88     NEWTRUE
#  356: u        SETITEMS   (MARK at 5)
#  357: .    STOP
#highest protocol among opcodes = 3
```

The most interesting piece of information in that disassembly is on the last line, because it includes the version of the pickle protocol with which this file was saved. There is no explicit version marker in the pickle protocol. To determine which protocol version was used to store a pickle file, you need to look at the markers (“opcodes”) within the pickled data and use hard-coded knowledge of which opcodes were introduced with each version of the pickle protocol. The `pickletools.dis()` function does exactly that, and it prints the result in the last line of the disassembly output. Here is a function that returns

just the version number, without printing anything:

```
import pickletools

def protocol_version(file_object):
    maxproto = -1
    for opcode, arg, pos in pickletools.genops(file_object):
        maxproto = max(maxproto, opcode.proto)
    return maxproto
```

And here it is in action:

```
import pickleversion
with open('entry.pickle', 'rb') as f:
    v = pickleversion.protocol_version(f)

print (v)
#3
```