26. v2/v3 Migration and Coexistence

Initial run of 2to3

`2to3` has a number of different modes of operation. The most useful way to get a handle on the likely difficulty of a conversion is to simply run it on the project's root directory. The command

```
2to3
.
```

scans all code ands report on suggested changes in the form of difference listings (diffs), summarizing which files likely need changing. Here is that summary:

```
RefactoringTool: Files that need to be
modified:
RefactoringTool:
./setup.py
RefactoringTool:
./tappio/__init__.py
RefactoringTool:
./tappio/lexer.py
RefactoringTool:
./tappio/models.py
RefactoringTool:
./tappio/parser.py
RefactoringTool:
./tappio/writer.py
RefactoringTool:
./tappio/scripts/extract.py
RefactoringTool:
./tappio/scripts/graph.py
RefactoringTool:
./tappio/scripts/indent.py
RefactoringTool:
./tappio/scripts/merge.py
RefactoringTool:
./tappio/scripts/missing_accounts.py
RefactoringTool:
./tappio/scripts/move_entries.py
RefactoringTool:
./tappio/scripts/print_accounts.py
RefactoringTool:
./tappio/scripts/print_earnings.py
RefactoringTool:
./tappio/scripts/renumber.py
RefactoringTool:
./tests/helpers.py
RefactoringTool:
./tests/script_import_test.py
RefactoringTool:
./tests/tappio_test.py
RefactoringTool:
./voitto/__init__.py
RefactoringTool:
./voitto/helpers/io.py
RefactoringTool:
./voitto/helpers/tracer.py
RefactoringTool: Warnings/messages while
refactoring:
RefactoringTool: ### In file ./tappio/scripts/missing_accounts.py
###
RefactoringTool: Line 36: You should use a for loop
here
```

The diffs show that approximately 20 source files are being recommended for change. As the old saying goes, why keep a dog and bark yourself? We can have `2to3` actually *make* the recommended changes by adding the `-w` flag to the command line. Normally `2to3` creates a backup file by renaming the original file from *.py* to *.py.bak*.

Operating under a version-control system renders this unnecessary, however, so we can also add the `-n` flag to inhibit the backups:

```
          2to3 -wn
(venv3) $ .
```

Here are the changes that `2to3` makes—we don't mention the filenames. Many of the changes are similar in nature, so we don't provide an exhaustive commentary on each one. As detailed next, many of the changes might be suboptimal; `2to3` is much more focused on producing working code than in optimizing its performance (remember the "golden rule of programming," covered in "Optimization"). Optimization is best treated as a separate step, once the code has been successfully converted and is passing its tests. The lines to be removed are preceded by minus signs (`-`), the replacement code by plus signs (`+`):

```
- assert type(ch) in (str,
unicode)
+ assert type(ch) in (str,
str)
```

v3, of course, does not define `unicode`, so `2to3` has converted it to `str` but failed to recognize that this no longer requires the tuple-membership test. Since the statement should work, however, leave this optimization for later.

```
- token =
self.token_iterator.next()
+ token =
next(self.token_iterator)
```

Several similar changes of this nature were recommended. In all cases the `next` method from v3, now renamed `__next__`, could simply transliterate the method call. However, `2to3` has chosen to produce better code, compatible across both versions, by using the `next` function instead.

```
- for account_number, cents in
balances.iteritems():
+ for account_number, cents in
balances.items():
```

Again several similar changes are recommended. In this case the converted code has a slightly different effect if run under v2, since the original was specifically coded to use an iterator rather than the list that (under v2) the `items` method produces. While this may use more memory, it shouldn't affect the result.

```
- map(all_accounts.update,
flat_accounts.itervalues())
+ list(map(all_accounts.update,
iter(flat_accounts.values())))
```

This is an interesting change because the original code was rather dubious—this is the line where the migration tool advised use of a `for` loop instead. Under v2, `map` returns a list while the v3 `map` is an iterator. Not only has the `itervalues` call been changed to `values` (because v3's `values` method produces an iterator; the `iter` call is

redundant, though innocuous), but a `list` call has also been placed around the result to ensure that a list is still produced. It might appear that the application of the `list` function serves no useful purpose, but removing the call would mean that the `all_accounts.update` function would only be applied to accounts as they were consumed from the iterator.

In fact the original code is somewhat dubious, and misused the `map` function. It would really have been little more complex, and far more comprehensible, to write

```
for ac_value in iter(flat_accounts.values()):
    all_accounts.update(ac_value)
```

as `2to3` advised. We leave this optimization for now, though, since despite its dubious nature the code as produced still works.

```
- print "{account_num}: ALL
{fmt_filenames}".format(**locals())
+ print("{account_num}: ALL
{fmt_filenames}".format(**locals()))
```

`2to3` ensures that all `print` statements are converted to function calls, and is good at making such changes.

```
- from StringIO import
StringIO
+ from io import
StringIO
```

The `StringIO` functionality has been moved to the `io` library in v3. Note that `2to3` does not necessarily produce code that is compatible across both versions; it pursues its principal purpose, conversion to v3—although in this case, as in previous ones, v2 is also happy with the converted code (since `StringIO` can also be accessed from `io` in v2, specifically in 2.7).

```
-
assert_true(callable(main))
+ assert_true(isinstance(main,
collections.Callable))
```

This change is a throwback because earlier v3 implementations removed the `callable` function. Since this was reinstated since v3.2, the conversion isn't really necessary, but it's harmless in both v2 and v3.

```
- except ParserError,
e:
+ except ParserError as
e:
```

This change also provides compatible syntax, good in both v2 and v3.

The `2to3` conversion detailed in this initial run of `2to3` does not, however, result in a working program. Running the tests still gives two errors, one in the `lexer_single` test and one in `writer_single`. Both complain about

differing token lists. This is the first (we've inserted line breaks to avoid too-wide code boxes running off the page):

```
...
  File "/Users/sholden/Projects/Python/voitto/tests/tappio_test.py",
       line 54, in lexer_single
    assert_equal(expected_tokens, tokens)
AssertionError: Lists differ: [('integer', '101')] !=
                              [<Token: integer '101'>]

First differing element 0:
('integer', '101')
<Token: integer '101'>

- [('integer', '101')]
+ [<Token: integer '101'>]
```

The second one is similar:

```
...
  File "/Users/sholden/Projects/Python/voitto/tests/tappio_test.py",
       line 188, in writer_single
    assert_equal(good_tokens, potentially_bad_tokens)
AssertionError: Lists differ:
  [<Tok[1169 chars]ce_close ''>, <Token: brace_close ''>,
   <Token: brace_close ''>] != [<Tok[1169 chars]ce_close ''>,
   <Token: brace_close ''>, <Token: brace_close ''>]

First differing element 0:
<Token: brace_open ''>
<Token: brace_open ''>

Diff is 1384 characters long. Set self.maxDiff to None to see it.
```

These were both traced to the token list comparison routine itself, which the original code provides in the form of a `__cmp__` method for the `Token` class, reading as follows:

```
def __cmp__(self, other):
    if isinstance(other, Token):
        return cmp((self.token_type, self.value),
                   (other.token_type, other.value))
    elif (isinstance(other, list) or
          isinstance(other, tuple)) and len(other) == 2:
        return cmp((self.token_type, self.value), other)
    else:
        raise TypeError('cannot compare Token to {!r}'.format(
                                                    type(
other)))
```

This method is not used in v3, which relies on so-called *rich comparison* methods. Since no `__eq__` is found, equality checks falls back to those inherited from `object`, which unfortunately don't measure up. The fix is to replace the `__cmp__` method with an `__eq__` method. This suffices to implement the equality checks required in the

tests; the new code is:

```
def __eq__(self, other):
    if isinstance(other, Token):
        return (self.token_type, self.value) ==
               (other.token_type, other.value)
    elif (isinstance(other, list) or
          isinstance(other, tuple)) and len(other) == 2:
        return (self.token_type, self.value) == tuple(other)
    else:
        raise TypeError('cannot compare Token to {!r}'.format(
                                                    type(
other)))
```

This final change is all that is required to have all tests pass under v3:

```
(ve35) airhead:voitto sholden$
nosetests
....
----------------------------------------------------------------------
Ran 4 tests in
0.028s

OK
```