How to Plan to Migrate to Python 3

Philip Semanchuk, PySpoken LLC, July 2018, PyOhio

About Me

- Longtime software developer
- Python user for ~10 years
- Freelancer! If you want to hire me: http://PySpoken.com
- Also an occasional photographer

Why Python 3, Why Now?

- Python 3 is mature;10 years old in December of this year (2018)
- Early versions of 3.x were not so great
- Python 3 has some appealing new features
- Python 2 will no longer be supported past 2019

A Plan for Your Plan

- Every organization is different
- This is an outline, you fill in the details
- Assumption: no simultaneous 2/3 support
- No matter who you are, **you can start immediately**!

Decide Which Python 3 to Target

- Use the most recent if possible

Some organizations have lots of choice as to which Python to use, some are more restricted. Examples --

- Red Hat® Enterprise Linux®, not so easy to get the latest Python

- A scientist who uses a cluster with an older Python 3.x installed

Reasons for Targeting the Latest

- Newer Pythons offer better performance
- Newer Pythons offer more features
- If you have to make a jump, may as well jump as far as you can to maximize the amount of time before your next jump
- Pythons ≤ 3.2 were smoothing off rough edges; definitely avoid those

Identify Test Gaps

- Is your test coverage 100%?
- Focus on text handling (especially around I/O)
- Focus on division as a source of sneaky bugs

Review Dependencies

- Review dependencies with `caniusepython3`

- It's sometimes too pessimistic

Start the Future Now (for New Code)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

- Enable Python 3 syntax in your Python 2 code
- Yes yes yes to this in *new* code
- Can break existing code

Start the Future Now (for Existing Code)

- `2to3` - utility, part of standard Python
- Consists of ~50 “fixers”
- `https://docs.python.org/2/library/2to3.html`

Start the Future Now — Three 2to3 Fixer Categories

1 . Now

2 . Now, With Review

3 . Later


CHER
DO NOT
MOVE
OK, I'LL JUST
STAND HERE

2to3 - Now

You can apply these immediately without breaking Python 2.7 compatibility.

1. apply
2. asserts
3. except
4. execfile
5. exitfunc
6. funcattrs
7. has_key
8. input
9. isinstance
10. methodattrs
11. ne
12. next
13. nonzero
14. numliterals
15. paren
16. reduce
17. repr
18. set_literal
19. standarderror
20. sys_exc
21. tuple_params
22. types
23. ws_comma
24. xreadlines

2to3 - Now

You can apply these immediately without breaking Python 2.7 compatibility.

1. apply
2. asserts
3. except
4. execfile
5. exitfunc
6. funcattrs
7. has_key
8. input
9. isinstance
10. methodattrs
11. ne
12. next
13. nonzero
14. numliterals
15. paren
16. reduce
17. repr
18. set_literal
19. standarderror
20. sys_exc
21. tuple_params
22. types
23. ws_comma
24. xreadlines

Now Fixer Example - ne

Not “Ni”, “ne”!

Converts the old not-equal syntax

<>

to

!=

2to3 - Now or Never

- `callable` fixer
- Specific to Python 3.1 support
- You almost certainly don't need the callable fixer

Now Fixers, With Review (and Testing)

Produce valid Python 2.7 code, but sometimes suboptimal choices.
Incorrect on rare occasions.

1. buffer
2. dict
3. filter
4. idioms
5. import
6. map
7. raise
8. renames
9. xrange
10. zip

Now Fixers, With Review (and Testing)

Produce valid Python 2.7 code, but sometimes suboptimal choices.
Incorrect on rare occasions.

1. buffer
2. dict
3. filter
4. idioms
5. import
6. map
7. raise
8. renames
9. xrange
10. zip

Example - The `dict` Fixer

Wraps the result of many dict methods with `list()`. Changes are conservative, harmless, and not always necessary.

Before (valid in both Python 2 and 3) –

```
for key in my_dict.keys():
    print(key)
```

After (valid in both Python 2 and 3) –

```
for key in list(my_dict.keys()):
    print(key)
```

Example - The `dict` Fixer Part 2

Wraps the result of many dict methods with `list()`. Changes are conservative, harmless, and not always necessary.

Before (valid in Python 2 only) –

```
foo = my_dict.keys() + ['a', 'b', 'c']
```

After (valid in both Python 2 and 3) –

```
foo = list(my_dict.keys()) + ['a', 'b', 'c']
```

2to3 - Later Fixers

- Python 2-incompatible changes
- Most very narrowly targeted (e.g. `getcwdu`), yawningly straightforward (`intern`), or both
- Unicode fixer deserves special attention

2to3 - Later Fixers

1. basestring
2. exec
3. future
4. getcwdu
5. imports and imports2
6. intern
7. itertools
8. itertools_imports
9. long
10. metaclass
11. print
12. raw_input
13. throw
14. unicode
15. urllib

2to3 - The Unicode Fixer

- First job - strip the 'u' prefix (meaningless in Python 3)
- Second job - convert `unicode()` to `str()` (this might break stuff)

- Solution - apply `@python_2_unicode_compatible` while still in Python 2 to make the effects of this fixer's changes predictable and consistent.

Ready, Set, Migrate!

- At this point you're well positioned

- Migrate to Python 3 as you see fit

Other Perspectives

Posted to reddit, didn't get love.

[-] **kankyo** 1 point 25 days ago

Seems like terrible advice to me. Going cleanly from 2 to 3 without passing six is fraught with danger and seems like a bad idea in almost all cases. You'd have to have a very small product, extremely good test coverage (over 90%), and a very small team for this to be a good idea.

permalink embed save report give gold **reply**

Other Perspectives

Things to note about the commenter's project --

- Sample size of 1
- 240k LOC excluding blank lines and comments
- Some of it ~15 years old
- https://medium.com/@boxed/moving-a-large-and-old-codebase-to-python3-33a5a13f8c99
- (Google ‘Hovmoller python3’)

Other Helpers

- *futurize* - “based heavily on 2to3”. Targets simultaneous 2/3 compatibility.

- *modernize* - “very thin wrapper around 2to3”, somewhat old. Targets simultaneous 2/3 compatibility via six.

Wrap Up

- Python 2 projects should migrate in the next 18-24 months
- Big projects need to start soon
- "Move fast and break things" vs. medical device company (or Hawaii Missile Alert System)
- `http://blog.pyspoken.com/2018/02/13/python-2-to-3-migration-guide/`

Thank You!

philip@PySpoken.com