Making WPT Python 3 Compatible

@stephenmcgruer, @svillar, @ziransun last-updated: 2020-03-31

Background

As of January 1st, 2020, Python 2 has been end-of-life'd. The Python maintainers will no longer perform bug fixes or even security fixes for Python 2 - the only supported version of Python is now Python 3.

The web-platform-tests (WPT) project uses Python in many places. These include, but are not limited to:

- The majority of our infrastructure code (e.g. the code underlying the `wpt` command); this primarily lives in tools/.
- WPT file handlers, which test authors can define to run custom code in response to them making a particular request to the WPT server.
- WebDriver tests, which use pytest structured tests.

This document summarizes the work we are doing to make WPT Python 3 compatible, and in particular the methodology we are taking.

As of 2020/03/03, this work is primarily being carried out by @svillar and @ziransun, with reviews and oversight provided by @stephenmcgruer, @jgraham, and the wider WPT Core team.

A Note on Test Coverage

Whilst porting from Python 2 to Python 3, it is a specific goal of this project to not reduce test coverage without explicit agreement from test authors. In general reducing test coverage is not expected to be needed, but in the case of Python-based test files there may be cases where a particular check is not needed under Python 3 and as such coverage is 'reduced'.

Major differences between Python 2 and Python 3

An entire book could be written on this subject, but the below are some of the main differences between Python 2 and Python 3. Each is marked up with the likely impact on the WPT codebase (a rough combination of complexity and severity).

Strings vs Bytes

Impact on WPT: High.

In Python 2, the string literal is called a "`str`" object but actually stores bytes. If you prefix it with "u" you get a "`unicode`" object that is stored as Unicode code points instead. In Python 3, a string literal is a sequence of Unicode code points instead, and you must prefix it with "b" to get bytes. [source]

Absolute imports

Impact on WPT: Low? (few ambiguous imports in WPT)

Absolute imports have become the default in Python 3. Relative imports should be explicit. [source]

built-in types comparison (Rules of ordering Comparisons):

Impact on WPT: Low (but it has happened!)

In Python 3: Most objects of built-in types compare unequal unless they are the same object. The choice of whether one object is smaller or larger than another one is made arbitrarily but consistently within one execution of a program. [source]

In Python 2 : In this case of 'mismatched' types, the types are listed lexicographically by type name: a "list" comes after an "int" in alphabetical ordering, so is greater. [source]

Iteration

Impact on WPT: Medium (mostly mechanical / easy to fix)

Python 3 changes the return values of several basic functions from `list` to `iterator`. The main reason for this change is that using iterators usually causes better memory consumption than lists. [source] This change has little impact on common use cases.

Furthermore, the iter* counterparts (which return iterators in Python 2) have been removed. We need to replace them with `six.iter*` to avoid memory regression in Python 2.

Integer division

Impact on WPT: Unknown, but suspected to be low.

Dividing two integers in Python 3 always results in a `float` value. In Python 2, dividing two integers always results in an integer (`int`) value. We need to be careful with this change when porting code, since the change in integer-division behavior can often go unnoticed (it doesn't raise a SyntaxError).

Leaking of variables

Impact on WPT: Unknown, but suspected to be low.

In Python 2, the value of the global variable will change while using it inside a for-loop. In Python 3, for-loop variables don't leak into the global namespace anymore. In [source], it is stated that:

“List comprehensions no longer support the syntactic form `[... for var in item1, item2, ...]`. Use `[... for var in (item1, item2, ...)]` instead. Also note that list comprehensions have different semantics: they are closer to syntactic sugar for a generator expression inside a list() constructor, and in particular the loop control variables are no longer leaked into the surrounding scope.”

Exceptions

Impact on WPT: Medium (purely mechanical / easy to fix).

In Python 3, an exception should be enclosed in parentheses. In Python 2, an exception should be enclosed in notations. See [source].

Print

Impact on WPT: Medium (purely mechanical / easy to fix).

in Python 3, the print statement has been replaced with a print () function. In Python 2, it is treated as a statement rather than a function.

Library dependencies

Impact on WPT: High (we have many dependencies).

Many older libraries created for Python 2 are not forward-compatible. And many recent developers are creating libraries which you can only use with Python 3. Tools such as caniusepython3 can be used to take in a set of dependencies and then figure out which of them are holding you up from porting to Python 3.

Methodology

There are two big challenges to making WPT compatible with Python 3; the dynamic quality of the language, and the nature of the changes between Python 2 and Python 3.

Starting with the latter; the differences between Python 2 and Python 3 are not just syntactical. If they were, we could just run some analyzer over the code and be done with. Instead, many of the changes are in the semantics - particularly the Strings vs Bytes described above - and as such whether or not Python 2 code is still correct in Python 3 depends on its purpose and the inputs to the function.

This complexity is compounded by the fact that Python is an interpreted language. Code paths can contain illegal semantics, but never be taken in 'normal' usage and thus escape even the reach of non-static analyzers. We can only be certain that we are Python 3 compatible when every code path has been traversed with every likely input in a Python 3 setting - an achievement that we expect to take years!

Starting with the important bits

WPT is a large codebase, and serves many auxiliary functions. Outside of '`wpt run`' (which itself does many things, as we'll see below), we have Python code for linting, interacting with docker, interacting with the CI systems, rebasing expectations, … the list goes on.

To make this tractable, we have decided to start with two very specific goals, each approaching the problem from different angles. Ziran (@ziransun) has been working on getting a basic '`wpt run`' command to execute under Python 3, whilst Sergio (@svillar) has been working on running all relevant unittests under Python 3. The former aims to get actual utility from the project (after all, our goal is running tests!) whilst the latter is targeting wider coverage via tests.

In both cases we are prioritizing within the tasks too. For example, to begin with we are reusing a Python 2 generated manifest (avoiding the manifest downloading or writing code), and utilizing a local browser binary rather than utilizing the built-in downloading code for those too. Similarly on the unittest side we are starting with the tests most related to core '`wpt run`' functionality, and skipping things like (again) manifest writing.

None of the above is to say we will leave part of WPT behind. Our goal is to make all of WPT compatible with both Python 2 and 3; we are just starting with the important bits!

The 'six' library

The 'six' library is a Python 2 and 3 compatibility library, which provides utility functions that hide minor differences between Python 2 and 3. For example, `HTMLParser` from Python 2 was moved to `html.parser` in Python 3; using six you can just write:

```
from six.moves import html_parser
```

Code coverage

Unit tests have had `pytest-cov` as dependency for some time. The `cov` plugin of pytest collects code coverage data from test runs. Code coverage is not run by default when running tests with `tox` but can be easily added.

The plan is to get a baseline of code coverage once we manage to get all unit tests running with python3. From that point the plan is to achieve full code coverage on critical paths and increase it in the others as much as possible. That would very likely imply adding more unit tests.

Static analysis

Python is a dynamic language, so static analysis is not that useful as with some other languages. However it can still provide some useful info in some cases. WPT is currently using `flake8` and `mypy`. The info we get from those tools is not generally useful for python3 migrations, except the static type analyses based on type hints.

Other analysis tools under consideration (but not currently used):

- pytype, which promises type checking without the need of annotations/hints.
- pylint, which can check if we deviate from Python 3 compatibility

There are some other tools out there like 2to3 or futurize that are useful for Python2 to Python3 migration, but run into the problem that we need to keep our codebase working with Python2 for the foreseeable future.

Current Status

Last updated: 2020/05/29

This section contains a high-level overview of the status of Python3 porting. It is not complete, instead just providing a 'flavour' of where we are.

'wpt run'

- All results below are currently using Chrome on Linux.
- 143 of top-level test directories in WPT run with identical results in both Python 2 and Python 3.

- 22 test directories rely on python file handlers, and porting these is on hold until we come up with a strategy for wptserve in Python 3.
- 14 test directories still have differences between Python 2 and Python 3, and are being investigated.

'wpt serve'

- We have identified that we need to be more strategic about bytes vs strings when it comes to wptserve and Python file handlers.
- Original/obsolete draft design with some discussions (Google-internal)
- Finalized/accepted RFC (exported from the design doc, note that the chosen approach was different from the originally recommended approach in the draft)
- Detailed guideline

WPT unittests

- Almost all known first-party unittests under `tools/`, `tools/wpt`, and `tools/wptrunner` now run both under Python 2 and Python 3.
- The exception is the tests for the http2 server, which currently does not work under Python 3 and which is considered low priority.

Code Coverage

- We now collect code coverage for all unittests in tools/.
- Work is underway on a way to easily extract and view the data (it is possible to do so today, but awkward).
- Code coverage data has been used to identify `` `wpt run --verify` `` as an undertested codepath, and tests are now being added for `stability.py`.