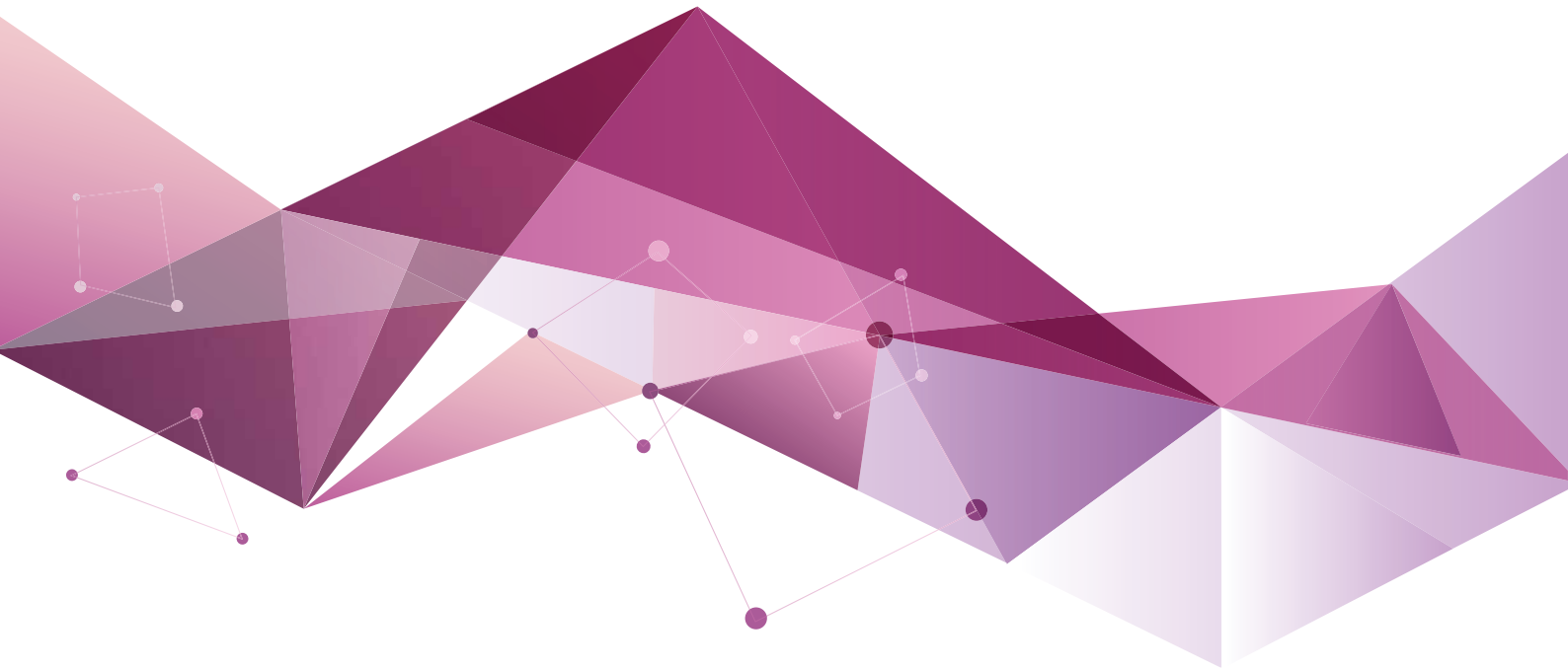
Suyati
dream. dare. do

PYTHON 3 AND DJANGO: WILL THE DUO WORK WELL TOGETHER?



Regardless of the compatibility issues with Django, is it the right time to shift to Python 3.x and tackle the issues head on?

django python

Suyati
dream. dare. do

Python, the popular object-oriented, interpreted, high-level programming language with dynamic semantics, got the new 3.0 version in 2008. This is when Guido van Rossum, the creator of Python, decided on a thorough cleanup of Python 2.x, fixing well-known annoyances and warts, and removing most of the old cruft.

Suyati
dream. dare. do

01
IMPROVEMENTS IN PYTHON 3.0

The major improvements in Python 3.0, also known as “Python 3000” or “Py3K” over Python 2.x are

- Better Unicode support. All text strings became Unicode by default, with saner bytes/Unicode separation.
- Major structural adjustments to several aspects of the core language, making it more consistent with the rest of the language and easier to learn. For instance, “Print” and “Exec” become statements and integers use floor division.
- Removal of old cruft. For example, all classes are now new-style, and “range()” returns a memory efficient iterable rather than a list as in 2.x.
- New approach to string formatting.

The problem was that the cleanup was done with scant regard for backward compatibility. As such, the Python 2.x development progressed simultaneously with Python 3.x, with Python 2.7 releasing in 2010. However, 2.7 is the final 2.x version. Python did offer extended support for this end-of-life release, but Python users looking to upgrade now may have no option but to make a move to Python

02

PYTHON 2.X OR PYTHON 3.X: WHICH TO USE?

In normal circumstances, it makes sense to shift to Python 3.0 for the simple reason that Python 2.x is legacy, and Python 3.x represents the present and future of this language. Python 3.x has had stable releases for more than five years, which include the 3.3 and 3.4 versions in the years 2012 and 2014, respectively. Moreover, the recent developments, including all the latest standard library improvements are only accessible by default in Python 3.x.

However, migrating from Python 2.7 to 3.x is not that straightforward or an open and shut case. In this case, a later version need not necessarily mean an advanced or even better version.

The reasons that inhibit migration to Python 3.x are:

1. Python 3.x still has inferior library support compared to Python 2.7, and most current Linux distributions and Macs still use Python 2.x as default.
2. Several new features of Python 3.x are incompatible with the older 2.x versions. The following are some features available only in 3.x releases and not back-ported to the 2.x series:
    - Default Unicode Strings
    - Clean Unicode/bytes separation
    - Exception chaining
    - Syntax for keyword-only arguments
    - Extended tuple unpacking
    - Non-local variable declarations
    - Function annotations

3. Most third-party software, especially in-house software used by businesses, still do not work in Python 3.x, or at least do not work properly. Also, popular modules such as Twisted, for networking and other applications, gevent, a network library, and others do not support Python 3.x. Retaining access to all functionality of such software require continued usage of Python 2.x.
4. Many resources, including the official tutorial for Django 1.6 are written for Python 2.X.
5. Python 3 is currently slower than Python 2.7.
6. Django, the popular web development framework for Python, is not yet fully compatible with Python 3.x.

Considering these factors, upgrading to Python 3.x is not a given, and depending on the exact circumstances of the business, it may make sense to continue with Python 2.x

Suyati
dream. dare. do

03 DJANGO

Django is a high-level Python web development framework. It offers a complete development environment that makes rapid development possible, while at the same time delivering a clean and pragmatic design. It is best suited to develop high-performing yet elegant web applications, like the ones used in newsrooms.

Some of Django's features are:

- A rich, dynamic, database access API. Django's potent Object-Relational Mapping system backs a huge set of database systems. Moving between engines is effortless, and requires only the changing of a configuration file.
- Automatic admin interface. Django offers ready-made and customizable admin interfaces, which spares developers the tedious task of creating interfaces to add and update content, and allows them to control Web applications through a Web portal easily.
- Elegant, cruft-free URL designs with no framework-specific limitations, offering the utmost in flexibility. Django's URLs are readable, boosting SEO.
- A powerful, extensible and designer-friendly template language that makes it possible to separate design, content and Python code. Django's powerful template system makes injecting programming logic inside templates easy, to simplify complex things.
- Granular cache system, with memcached or other cache frameworks offering super performance.
- Ultra lightweight web server for development and testing. The debugging mode offers very thorough and detailed error messages, making it possible to isolate bugs very easily.

Suyati
dream. dare. do

04
DJANGO AND PYTHON 3.0

Django is based on Python 2.7. Since the launch of Python 3.0, Django, along with other major Python based packages such as Flask, CherryPy, Pyramid, NumPy, cx_Freeze, and py2exe are in the process of being ported to Python 3.x.

Django 1.5 was the first version of Django to support Python 3. This, however, offered only "experimental" support. Official support for Python 3.0 came in Django 1.6, and the September 2014 launch of Django 1.7 makes the support better.

However, Django, even the latest 1.7 version, is by no means ready for Python 3. The porting is not complete, and running Django with Python 3.0 could still pose some problems.

Even as Django 1.7 has theoretically resolved most of the compatibility issues that plagued earlier versions, there are still several known issues. One such is the inability to migrate database with Django 1.7. Moreover, it has still not received real-life testing under Python 3. While the core development team did its best to eliminate bugs, the test suite does not cover all possible uses of Django.

Meanwhile, the release of Django 1.7 has led to Django 1.5 reaching its end-of-life. Django 1.6 will continue to receive support until the release of Django 1.8, expected in March 2015. However, none of these releases – Django 1.5, 1.6 or 1.7 are fully stable. Django 1.4 is the long-term support release, and this does not support Python 3.0.

Suyati
dream. dare. do

05
PROBLEMS OF RUNNING DJANGO WITH PYTHON 3.X

Django 1.5 and below contain several string related classes and functions in their “django.utils.encoding” and “django.utils.safestring” modules. Many of these classes and functions use “str” and “Unicode”. Python 3 considers all strings as Unicode by default. The “Unicode” type in Python 2 is renamed “str” in Python 3, and “str()” in Python 2x is renamed “bytes”. As such, this creates all round confusion.

Adding to the confusion is Python 3 not having automatic conversions between str and bytes, and the codecs module becoming more strict. In Python 3, str.encode() always returns bytes, and bytes.decode always returns “str”.

Python 2’s “u” prefix shows up as a syntax error in Python 3.2 but is allowed in Python 3.3. In Python 3.2, it becomes necessary to prefix byte strings with “b”.

06 BACKWARD COMPATIBILITY

Python 2.6+ and Python 3.3+ share a fairly large common subset. This, combined with the restoration of “u” prefix support for Unicode literals in Python 3.3 makes it possible to make semantically correct Python 2.6+ code source compatible with Python 3.3+. The main task is to import some things from different places to resolve the different naming in Python 2 and Python 3.

The DjangoCon Europe sprints took up the task of making Python 3 backward compatible, to enable it to work seamlessly with older versions of Django. The core team wrote Python 3 code with Python 2 compatibility, rather than the other way round, to ensure that the codebase remains future-proof.

The major work-through were:

- Renaming “string” as “bytes” and “Unicode” as “text” in functions and classes to remove the confusion caused by these terms meaning two different things in Python 2 and Python 3.
- Correcting several approximations that lingered from previous versions. Python 3 requires string encoding and decoding operations to be correct. This had a spin-off benefit of cleaning up the code considerably.
- Making changes in Unicode literals, such as removing the “u” prefix before Unicode strings, adding a “b” prefix before bytestrings, and adding “from __future__ import unicode_literals” at the top of Python modules.

Based on these efforts, it is now possible to write a compatible source code in Django 1.5 to address much of these issues using the Python 2/3 Compatible Source strategy.

The “six” compatibility package is the key utility for supporting Python 2 and Python 3 in a single code base. “Six” is a single Python file that offers utilities for wrapping over differences between Python 2 and Python 3, supporting codebases

to work on both Python 2 and 3 without modification. Invoking django.utils.six, a compatibility tool offered by Django 1.5 makes it possible to run the same Django code on both Python 2.6.5 or above and Python 3.2 or above. This tool is a customized version of Python's "six" compatibility layer.

Using this compatibility code, it is possible to make Python 3 backward-compatible, which is making the code compatible with Python 2. It is not possible to make Python 2 code compatible with Python 3.

Based on these revisions, the strings and classes were renamed in Django's "django.utils.encoding" and "django.utils.safestring" modules. For instance, in django.utils.encoding. "smart_str" becomes "smart_bytes", "smart_unicode" becomes "smart_text" and "force_unicode" becomes "force_text". In django. utils.safestring, the mark_safe() and mark_for_escaping() functions did not change, but "EscapeString" became "EscapeBytes", "Escape Unicode" became "EscapeText", "SafeString" became "SafeBytes", and "SafeUnicode" became "SafeText".

Also, bytestrings and Unicode strings that contain only ASCII data are interchangeable in Python 2, and as such one can use Unicode strings everywhere and not use the "b" prefixes. However, writing such a compatible source code can be frustrating.

All things considered, it may make sense to persist with Python 2.0 for some more time. However, the onus should be on writing good code. A well-written 2.x code can be a lot like 3.x code, and 2.x v/s 3.x becomes less of an issue. That can mean many things, including using new-style classes, not using ancient deprecated incantations of print, and using lazy iterators where available.

Suyati
dream. dare. do

Suyati Technologies

Suyati is a young, upwardly mobile company focused on delivering niche IT services to support myriad Digital Engagement strategies. Our expertise also includes integration and delivery of CRM, CMS and e-commerce solutions.

www.suyati.com **services@suyati.com**

Reference:

1. *http://nick-coghlans-python-notes.readthedocs.org/en/latest/*
2. *http://stackoverflow.com/questions/26298723/is-django-faster-on-python-3*
3. *http://stackoverflow.com/questions/8945938/django-compatibility-issues-with-python-3-2*
4. *http://www.opensourceforu.com/2013/03/django-a-web-framework-with-immense-potential/*
5. *http://www.pydanny.com/experiences-with-django-python3.html*
6. *http://www.screamingatmyscreen.com/2012/2/business-decision-why-i-use-django-and-not-ruby-on-rails/*
7. *https://docs.djangoproject.com/en/1.7/topics/python3/*
8. *https://docs.python.org/3.5/whatsnew/index.html*
9. *https://github.com/bread-and-pepper/django-userena/issues/398*
10. *https://wiki.python.org/moin/Python2orPython3*
11. *https://www.djangoproject.com/weblog/2012/aug/19/experimental-python-3-support/*