Support
Python 2 *and* Python 3
with the same code

EuroPython 2014

Stefan Schwarzer, SSchwarzer.com
info@sschwarzer.com

Berlin, Germany, 2014-07-24

Introduction

About me

- Degree in chemical engineering
- Software developer since 2000
- Freelancer since 2005
- Maintainer of ftputil, an FTP client library for Python
- Last year release of ftputil 3.0, with support for Python 3.x in addition to Python 2.6 and 2.7. Same source code, same API.

Python 2 or Python 3?

Should I use Python 2 or Python 3 for my development activity?

. . .

Python 2.x is legacy, Python 3.x is the present and future of the language.

https://wiki.python.org/moin/Python2orPython3

So Python 2 is obsolete?

- Yes. And no. ;-)
- Python 2 still more widely used
- Python 2 pre-installed on many Linux distributions, Python 3 is optional.
- More hosting for Python 2
- Many libraries don't have a Python 3 version yet. (But many do.)
- That said, use Python 3 if you can :-)
- Ease transition for others by providing Python 3 support in your libraries

☺

Different approaches

- Develop in Python 2, make Python 3 version with `2to3`
- Develop in Python 3, make Python 2 version with `3to2` (rare)
- Develop in Python 2 and Python 3
  - Same source code
  - No conversion step during installation
  - . . . and development

Bytes vs. unicode

Bytes vs. unicode

- Bytes (= byte strings) represent bytes as stored on disk or sent over a socket
- Unicode (= unicode text, unicode strings) represents character data. Characters have number codes ("code points") that are unrelated to how the characters are stored
- Unicode text can be encoded to bytes according to an encoding (e. g. UTF-8 or Latin1)
- Conversely, bytes can be decoded to unicode text

68	h		68
f6	ö	encode	c3
		$\longrightarrow$	b6
72	r	$\longleftarrow$	72
65	e	decode	65
6e	n		6e

Python string types

Python	Binary type	Unicode type	Default string literal type, as in `"string"`
2	`str`	`unicode`	`str` (= binary type)
3	`bytes`	`str`	`str` (= unicode type)

Python 3: Unicode and byte strings can't be mixed

- Valid code in Python 2:
  `s = "a sequence of bytes" + u" and unicode text"`
- Unless the bytes sequence contains anything non-ASCII:
  `UnicodeDecodeError` – dependent on processed data
- In Python 3, mixing bytes and unicode like this isn't allowed:
  `TypeError` – independent of processed data
- In Python 3, decode bytes or encode the unicode text to combine the strings:
  `s = b"bytes".decode("utf8") + " and unicode"`
  However, don't do this "in-place" like here

Python 3: Most APIs use unicode instead of byte strings

```
>>> import decimal
>>> decimal.Decimal("1.0")  # Python 3 unicode string
Decimal('1.0')
>>> decimal.Decimal(b"1.0")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: conversion from bytes to Decimal is not supported
```

Python 3: New file API

- `open()` creates files for text or binary data (`t` and `b`, as usual)
- Text files accept only unicode for writing and return unicode on reading
- Binary files accept only bytes for writing and return bytes on reading
- No more `file` objects. Return value of `open` depends on the arguments (unicode vs. bytes, buffered vs. unbuffered).
- New `open` function also available as `io.open` in Python 2.6, 2.7 and 3.x

Python 3: Operating system interfaces use unicode

- Command line arguments: `sys.argv` contains unicode strings
- Standard input: `sys.stdin.read()` returns unicode string
- Standard output: `sys.stdout.write(string)` requires unicode argument
- `sys.stdin.buffer` and `sys.stdout.buffer` to read or write binary data

Adapting for Python 3

Steps

Have/write automated unit tests

- If you don't have them, this is a good time to write them
- Must pass 100 % under Python 2 (and later Python 3)
- Adapt/add/delete tests during changes for Python 3 support
- Lots of tests will fail if run under Python 3 for the first time. That's normal. Don't despair! :-)
- Running tests with `python -3` prints information on things that likely need to change for Python 3 support
- Frequently make sure production code and tests are in sync
- Run tests under both Python 2 and 3 with `tox`

Run `2to3` once

Makes some simple, but helpful changes

- `print` is now a function
  `print("Answer:", 42, file=results_file)`
- Exceptions
  `except SomeError as error:`
- . . . and other changes
- Look up `2to3` documentation on "fixers"
- Exclude `future` fixer
- Some fixer changes shouldn't be kept as-is. Replace them with code that will work with Python 2 and 3.

Inspect results thoroughly and fix any problems.

After this all unit tests should pass (again) under Python 2.

Change APIs to support Python 2 and 3

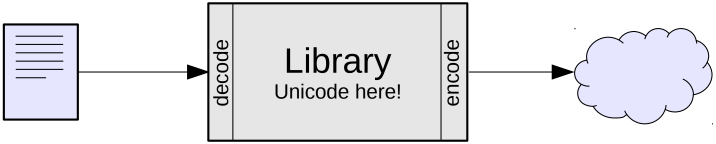
Unicode or bytes for text?

- Standard library of Python 2 accepts both unicode and bytes
- Standard library of Python 3 accepts almost only unicode (exception: file system paths)
- Therefore use unicode for text data
- Know/define what data is text data (e. g. are URLs binary or text?)

Change APIs to support Python 2 and 3

Encode/decode text at system boundaries

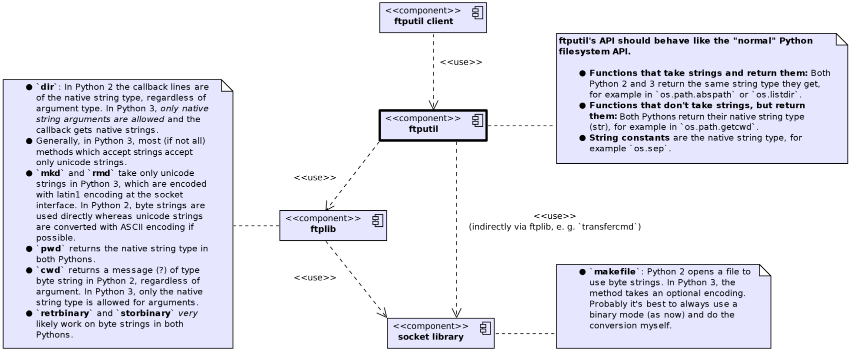
- Decode bytes to text as soon as possible (e. g. `io.open` with `encoding` argument to read text data from a file)
- Encode text data to bytes as late as possible (e. g. before sending text data over a socket)
- Hence all text "inside" the system is unicode



Change APIs to support Python 2 and 3

If the API changes aren't straighforward

Visualize your system. Pay attention to components and interactions.



<<component>> ftputil client
<<use>>
<<component>> ftputil
<<use>>
<<component>> ftplib
<<use>>
<<use>> (indirectly via ftplib, e. g. `transfercmd`)
<<component>> socket library

- `**dir**`: In Python 2 the callback lines are of the native string type, regardless of argument type. In Python 3, *only native string arguments are allowed* and the callback gets native strings.
- Generally, in Python 3, most (if not all) methods which accept strings accept only unicode strings.
- `**mkd**` and `**rmd**` take only unicode strings in Python 3, which are encoded with latin1 encoding at the socket interface. In Python 2, byte strings are used directly whereas unicode strings are converted with ASCII encoding if possible.
- `**pwd**` returns the native string type in both Pythons.
- `**cwd**` returns a message (?) of type byte string in Python 2, regardless of argument. In Python 3, only the native string type is allowed for arguments.
- `**retrbinary**` and `**storbinary**` *very* likely work on byte strings in both Pythons.

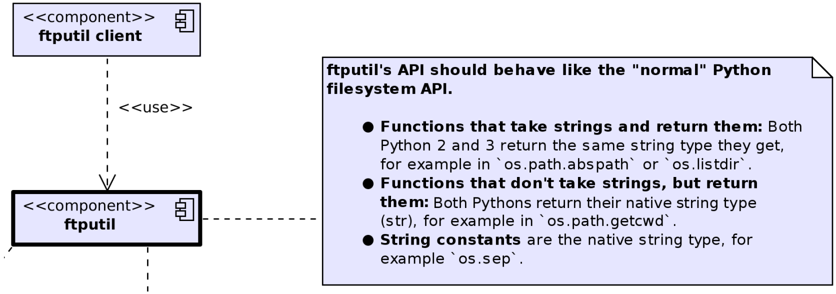
ftputil's API should behave like the "normal" Python filesystem API.

- **Functions that take strings and return them:** Both Python 2 and 3 return the same string type they get, for example in `os.path.abspath` or `os.listdir`.
- **Functions that don't take strings, but return them:** Both Pythons return their native string type (str), for example in `os.path.getcwd`.
- **String constants** are the native string type, for example `os.sep`.

- `**makefile**`: Python 2 opens a file to use byte strings. In Python 3, the method takes an optional encoding. Probably it's best to always use a binary mode (as now) and do the conversion myself.

Change APIs to support Python 2 and 3

If the API changes aren't straighforward

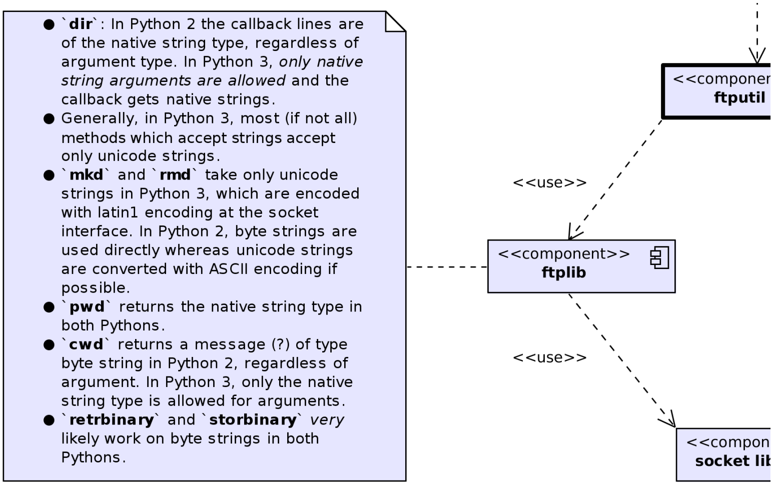


ftputil's API should behave like the "normal" Python filesystem API.

- **Functions that take strings and return them:** Both Python 2 and 3 return the same string type they get, for example in `os.path.abspath` or `os.listdir`.
- **Functions that don't take strings, but return them:** Both Pythons return their native string type (str), for example in `os.path.getcwd`.
- **String constants** are the native string type, for example `os.sep`.

Change APIs to support Python 2 and 3

If the API changes aren't straighforward



- `**dir**`: In Python 2 the callback lines are of the native string type, regardless of argument type. In Python 3, *only native string arguments are allowed* and the callback gets native strings.
- Generally, in Python 3, most (if not all) methods which accept strings accept only unicode strings.
- `**mkd**` and `**rmd**` take only unicode strings in Python 3, which are encoded with latin1 encoding at the socket interface. In Python 2, byte strings are used directly whereas unicode strings are converted with ASCII encoding if possible.
- `**pwd**` returns the native string type in both Pythons.
- `**cwd**` returns a message (?) of type byte string in Python 2, regardless of argument. In Python 3, only the native string type is allowed for arguments.
- `**retrbinary**` and `**storbinary**` *very* likely work on byte strings in both Pythons.

Change APIs to support Python 2 and 3
Some tips

- Don’t let functions/methods accept both unicode and byte strings → makes API confusing and complicates tests
- Special case: prefer file-like objects over strings for paths
- Avoid different APIs for Python 2 and 3 → complicates other code that should work under Python 2 and 3
- Make a list of changes before actually changing the API
  - Notice more easily if something is missing before coding
  - Makes it more difficult to forget some changes during coding
  - Helps write release notes and possibly ”What’s new?“ document
- Increasing major version number justifies necessary backward-incompatible API changes :-)

Adapting for Python 3

Tips

Read "What's new in Python 3.0 ?“

At least.

Highly recommended: "Porting to Python 3"

If possible, support only Python 2.6 and up

- Python 2.6 and 2.7 have very useful Python 3 features backported, e. g. the `print` function, a new exception syntax and the `io` module
- Use `six` library to support Python 2.5
- Anything below 2.6 will probably be awkward

More tips

- `compat` module for things that need to differ between Python 2 and 3

```
if sys.version_info[0] == 2:
    ...
else:
    ...
```

- Consider `future` or `six` library for larger projects
- Add to every Python file

```
from __future__ import (absolute_imports,
  division, print_function, unicode_literals)
```

- Alternative to `unicode_literals` in Python 3.3+ : `u` prefix for unicode strings

Summary

Summary

- Python 2 is in wider use, but Python 3 is the future
- Using the same source code to support Python 2 and 3 is feasible
- Know the concepts of unicode, bytes and encodings and the related changes from Python 2 to Python 3
- Without unit tests, adapting for Python 3 will usually be much more difficult
- Prefer text APIs in "Python 3 style"
- Plan and implement necessary API changes carefully
- Read "What's new in Python 3.0?" for more differences
- Python 2.6 and 2.7 have many features backported from Python 3, so require at least Python 2.6 if possible

Thanks for your attention! :-)

Questions?

Remarks?

Discussion?

sschwarzer@sschwarzer.com

http://sschwarzer.com

Links

- Unicode article by Joel Spolsky
  http://www.joelonsoftware.com/articles/Unicode.html
- Python 3 resources
  http://getpython3.com
- Porting to Python 3 (free online book)
  http://python3porting.com
- Modernize
  https://github.com/mitsuhiko/python-modernize
- future and six libraries
  http://python-future.org
  https://pypi.python.org/pypi/six
- tox
  http://tox.readthedocs.org