Why is it so difficult to retire an unused function?

Gavin Chan

Principal Quant Developer, AXA IM Chorus Ltd

My Background

- AXA IM Chorus Ltd is a quantitative asset management fund.
- Develop in Python
- Research in Python
- Manage the portfolio in Python

Story

[Why computers suck and how learning from OpenBSD can make them marginally less horrible]

- Enterprise oriented development model - ABI backward compatibility is prioritized and access to deprecated APIs remains between major releases

- Non-enterprise development, e.g. OpenBSD, do not hesitate to introduce ABI break changes for code quality
- OpenBSD considers a lack of documentation on any function of the system to be a bug

Myth

- Lack of diligence and attitude?
- Lack of incentive?
- Lack of development time and resource?

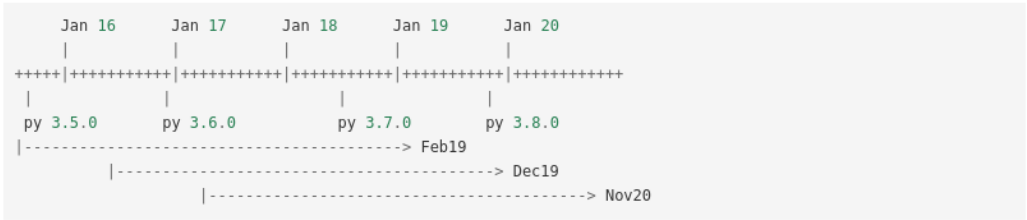
Hard to get the stains off?

- Uncertainty and undefinite business value to clean technical debt
- Unclear usage from the clients and downstream processes on the existing application
- Missing tools / systematic approach to remove unused codes

Approach - Planning and Warning

- Alerts users and developers on the deprecation plan
- Requires planning ahead on the support timeline
- Documentation / Enhancement proposal can help draw the timeline

Consider the following timeline:



It shows the 42 month support windows for Python. A project with a major or minor version release in February 2019 should support Python 3.5 and newer, a project with a major or minor version released in December 2019 should support Python 3.6 and newer, and a project with a major or minor version release in November 2020 should support Python 3.7 and newer.

- Semantic Versioning - MAJOR version when you make incompatible API changes

Approach - Expired and Cleaning

- Expired stage throws an exception on the application level if the function is called
- Cleaning stage removes the deprecation part from the source code

Auto-deprecator

What happened when the function should be deprecated in version 2.0.0?
@deprecate(expiry='2.0.0', ...)

Version	1.9.0	2.0.0	2.1.0
Stage	**Warning**	**Expired**	**Cleaning**
User	Deprecation warning alerts the users the expiry version of the deprecating function	Exception is thrown out with message that the function is deprecated	The function is completely removed from the source code
Dev	Unit / Integration Test via injecting environment variable DEPRECATED_VERSION	Production support can still rollback by hijacking the current version, i.e. environment variable "DEPRECATED_VERSION"	Release process can automatically remove the deprecated function from the source code before the release

Auto-deprecator - Warning

- Specify the current version and the target expired version

- Specify the current version by the package version and the migrated function

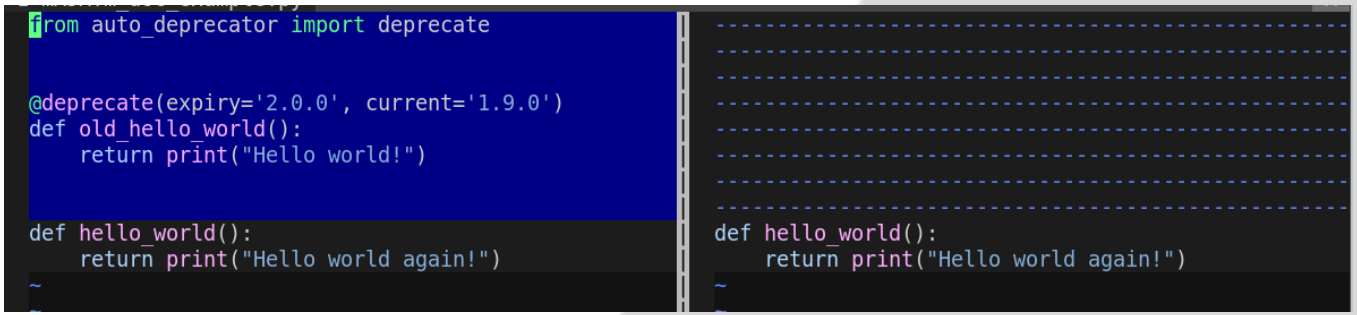
Auto-deprecator - Warning and Expired

- User case

- Downstream test case

Auto-deprecator - Cleaning

- Command removes the deprecated function from the source code



Frequent rewrites

"Most software at Google gets rewritten every few years"

"Software that is a few years old was designed around an older set of requirements and is typically not designed in a way that is optimal for current requirements."

"Rewriting code cuts away all the unnecessary accumulated complexity that was addressing requirements which are no longer so important"

If you are interested to the project **auto-deprecator**, please visit
https://github.com/auto-deprecator/auto-deprecator

Github: @gavincyi

Email: gavincyi@gmail.com

Linkedin: Gavin, Ying In Chan

AXA Investment Managers