python-consistency Documentation

Release

Ralph Broenink

Jun 11, 2017

General

I love Python. But there's one thing that bothers me: inconsistent naming in many modules in its standard Library.

Some names are surprising, inconsistent across modules, or simply incorrect. This is mostly caused by the fact tha several modules were developed before the introduction of PEP-8, and now we're stuck with these names in older modules.

It has been said and discussed in the past that the stdlib is in fact inconsistent, but fixing this has almost always been disregarded as being too painful (after all, we don't want a new Python 3 all over again). However, this way, we will never move away from these inconsistencies. Perhaps this is fine, but I believe that with some effort, we can fix this for generations to come.

This module was written based on a discussion on python-ideas I started in July 2016 as an attempt to get this fixed for once and for all. Although the core developers don't see a need to fix this at this point in time, as it requires a lot of effort that is simply not worth the benefits, I still feel it should be part of Python's future.

While maintaining full backwards compatibility, this module adds consistently named aliases to modules in the standard library (as suggested in the linked thread). This module currently is nothing more than a bunch of renames that you can import. For instance:

```
from consistency import logging

logging.logger(__name__)
```

Ultimately, I feel that Python itself should provide these properly named alternatives. The original variant should be aliased with them (or the other way around), without defining a deprecation timeline for the original names. This should make it possible to eventually make the stdlib consistent, Pythonic and unsurprising.

CHAPTER 1

Naming Conventions

This module uses several naming conventions. These conventions are invented, as PEP-8 only specifies how you should format your names (e.g. snake_case) and not how you should actually pick your names.

We refer you to PEP-8 for naming styles, such as when to use CamelCase or snake_case. There are actually many violations inside the standard library for this simple convention, e.g. `unittest` and `logging` for CamelCased function names, and `collections` and `datetime` for lowercased class names.

Naming

- **Consistency across modules**

  Modules should be consistent with each other, e.g. `tarfile.TarFile.add` and `zipfile.ZipFile.write` are inconsistent.

- **Underscores between words**

  There should be underscores between different English words, e.g. `http.client.HTTPConnection.getresponse` is wrong. There are some exceptions listed below.

- **American English**

  A quick survey found that most of Python is currently in American English, so we prefer this.

Accepted single-worded names

The following words are accepted as single words, although the dictionary may say otherwise. However, this also means that we always must see these words in the form listed in this table. There's no 'sometimes' here.

Name	Reason
filename	Commonly used and interpreted as single word.

Properties

- **Properties in favor of getters/setters**

  Getters and setters that are simple, e.g. no parameters in the getter and a single in the setter, should be a property. However, if there are significant side effects to the getter or the setter, that must be made clear to the programmer, use the function style.

- **Do not use** `get_`

  Prefer to use the name without the `get_` prefix. This is in line with the use of properties, but also when it is a method, prefer to use it without the prefix. Unless you also have a setter, but then you would have used a property anyway. Conversely, `set_` should be avoided as well, but only if this is clear.

- `is__` **is a property**

  If you have a method that is simply a `def is_foo(self):`, it is a property with that name.

- **Prefer using iterators, avoid** `iter_`

  Unless you need to distinguish between iterators and lists, you should avoid the prefix `iter_`. Furthermore, if your code of returning a list is simply `list(iter)`, avoid that method at all. But if you have a list, return it. A list is iterable after all.

Files

- **Avoid many methods for working on strings and bytes and file-like objects**

  Having four methods for working on a set of different inputs really does not look very nice. Python 3.4 introduced the notion of single-dispatch generic functions (see PEP-443), so we should use those.

- **Avoid writing to a file directly**

  Avoid writing your output to a file. You need the `io` library to get your raw output. If you have useful optimizations by writing to a file instead of to a string, at least make it an option.

This module

- **Low-level modules that have a higher level module are not renamed**

  We do not provide renames for modules that are low-level and a higher level exists. This includes, for instance, the `os.path` module, as you you should be using `pathlib` anyway.

- **Superseded modules are not renamed**

  We do not provide renames for obsolete modules, such as `optparse`.

- **Builtins are not renamed**

  For now. We spotted the forbiddenfruit module, so there's still hope.

CHAPTER 2

logging

Warning: The logging module is intended to be subclassed. We currently do not provide a way for this. We are working on it.

Note: Only those documented in section 16.6 have been done, logging.handlers and logging.config are still to be done.

Renames

The following renames have been made:

Logger

Previous name	New name	Rationale
`setLevel`	`set_level`	CamelCasing
`isEnabledFor`	`is_enabled_for`	CamelCasing
`getEffectiveLevel`	`effective_level` (property)	CamelCasing and getter should be property
`getChild`	`child`	CamelCasing
`addFilter`	`add_filter`	CamelCasing
`removeFilter`	`remove_filter`	CamelCasing
`addHandler`	`add_handler`	CamelCasing
`removeHandler`	`remove_handler`	CamelCasing
`findCaller`	`find_caller`	CamelCasing
`makeRecord`	`make_record`	CamelCasing
`hasHandlers`	`has_handlers` (property)	CamelCasing and getter should be property

Handler

Previous name	New name	Rationale
`createLock`	`create_lock`	CamelCasing
`setLevel` / `level`	`level` (property) (and `_level`)	setLevel was a setter for the already existing level property
`setFormatter`	`formatter`	setFormatter did only change existing property
`addFilter`	`add_filter`	CamelCasing
`removeFilter`	`remove_filter`	CamelCasing
`addHandler`	`add_handler`	CamelCasing
`handleError`	`handle_error`	CamelCasing

Formatter

Previous name	New name	Rationale
`formatTime`	`format_time`	CamelCasing
`formatException`	`format_exception`	CamelCasing
`formatStack`	`format_stack`	CamelCasing

Module-level

Note: This still needs a lot of work, as we actually want to change the way the getters and setters work. Also, last_resort does not work properly yet.

Previous name	New name	Rationale
`getLogger`	`logger`	CamelCasing and losing get as this is implied by the name
`getLoggerClass`	`get_logger_class`	CamelCasing and since this is module-level, doing properties would be too hard
`setLoggerClass`	`set_logger_class`	see above
`getLogRecordFactory`	`get_log_record_factory`	see above
`setLogRecordFactory`	`set_log_record_factory`	see above
`addLevelName`	`add_level_name`	CamelCasing
`getLevelName`	`get_level_name`	CamelCasing
`makeLogRecord`	`make_log_record`	CamelCasing
`basicConfig`	`basic_config`	CamelCasing
`lastResort`	`last_resort`	CamelCasing
`captureWarnings`	`capture_warnings`	CamelCasing

CHAPTER 3

Indices and tables

- genindex
- modindex
- search