Pylint Documentation

Release 1.9.5

Logilab, PyCQA and contributors

Jul 16, 2019

Contents

CHAPTER 1

Introduction

1.1 What is Pylint?

Pylint is a tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells. It can also look for certain type errors, it can recommend suggestions about how particular blocks can be refactored and can offer you details about the code's complexity.

Other similar projects would include the now defunct pychecker, pyflakes, flake8 and mypy. The default coding style used by Pylint is close to PEP 008.

Pylint will display a number of messages as it analyzes the code and it can also be used for displaying some statistics about the number of warnings and errors found in different files. The messages are classified under various categories such as errors and warnings.

Last but not least, the code is given an overall mark, based on the number and severity of the warnings and errors.

1.2 What Pylint is not?

What Pylint says is not to be taken as gospel and Pylint isn't smarter than you are: it may warn you about things that you have conscientiously done.

Pylint tries hard to report as few false positives as possible for errors, but it may be too verbose with warnings. That's for example because it tries to detect things that may be dangerous in a context, but are not in others, or because it checks for some things that you don't care about. Generally, you shouldn't expect Pylint to be totally quiet about your code, so don't necessarily be alarmed if it gives you a hell lot of messages for your project!

The best way to tackle pylint's verboseness is to:

- enable or disable the messages or message categories that you want to be activated or not for when pylint is analyzing your code. This can be done easily through a command line flag. For instance, disabling all convention messages is simple as a `--disable=C` option added to pylint command.
- create a custom configuration file, tailored to your needs. You can generate one using pylint's command `--generate-rcfile`.

CHAPTER 2

Tutorial

Author Robert Kirkpatrick

2.1 Intro

Beginner to coding standards? Pylint can be your guide to reveal what's really going on behind the scenes and help you to become a more aware programmer.

Sharing code is a rewarding endeavor. Putting your code `out there` can be either an act of philanthropy, `coming of age`, or a basic extension of belief in open source. Whatever the motivation, your good intentions may not have the desired outcome if people find your code hard to use or understand. The Python community has formalized some recommended programming styles to help everyone write code in a common, agreed-upon style that makes the most sense for shared code. This style is captured in PEP-8. Pylint can be a quick and easy way of seeing if your code has captured the essence of PEP-8 and is therefore `friendly` to other potential users.

Perhaps you're not ready to share your code but you'd like to learn a bit more about writing better code and don't know where to start. Pylint can tell you where you may have run astray and point you in the direction to figure out what you have done and how to do better.

This tutorial is all about approaching coding standards with little or no knowledge of in-depth programming or the code standards themselves. It's the equivalent of skipping the manual and jumping right in.

My command line prompt for these examples is:

```
robertk01 Desktop$
```

2.2 Getting Started

Running Pylint with no arguments will invoke the help dialogue and give you an idea of the arguments available to you. Do that now, i.e.:

```
robertk01 Desktop$ pylint
...
a bunch of stuff
...
```

A couple of the options that we'll focus on here are:

```
Master:
  --generate-rcfile=<file>
Commands:
  --help-msg=<msg-id>
Commands:
  --help-msg=<msg-id>
Message control:
  --disable=<msg-ids>
Reports:
  --files-output=<y_or_n>
  --reports=<y_or_n>
  --output-format=<format>
```

Also pay attention to the last bit of help output. This gives you a hint of what Pylint is going to `pick on`:

```
Output:
   Using the default text output, the message format is :
  MESSAGE_TYPE: LINE_NUM:[OBJECT:] MESSAGE
  There are 5 kind of message types :
  * (C) convention, for programming standard violation
  * (R) refactor, for bad code smell
  * (W) warning, for python specific problems
  * (E) error, for much probably bugs in the code
  * (F) fatal, if an error occurred which prevented pylint from doing
  further processing.
```

When Pylint is first run on a fresh piece of code, a common complaint is that it is too `noisy`. The current default configuration is set to enforce all possible warnings. We'll use some of the options I noted above to make it suit your preferences a bit better (and thus make it emit messages only when needed).

2.3 Your First Pylint'ing

We'll use a basic python script as fodder for our tutorial. I borrowed extensively from the code here: http://www.daniweb.com/code/snippet748.html The starting code we will use is called simplecaeser.py and is here in its entirety:

```
 1  #!/usr/bin/env python
 2
 3  import string
 4
 5  shift = 3
 6  choice = raw_input("would you like to encode or decode?")
 7  word = (raw_input("Please enter text"))
 8  letters = string.ascii_letters + string.punctuation + string.digits
 9  encoded = ''
10  if choice == "encode":
11      for letter in word:
12          if letter == ' ':
13              encoded = encoded + ' '
```

(continues on next page)

(continued from previous page)

```
14          else:
15              x = letters.index(letter) + shift
16              encoded=encoded + letters[x]
17  if choice == "decode":
18      for letter in word:
19          if letter == ' ':
20              encoded = encoded + ' '
21          else:
22              x = letters.index(letter) - shift
23              encoded = encoded + letters[x]
24
25  print encoded
```

Let's get started.

If we run this:

```
robertk01 Desktop$ pylint simplecaeser.py
No config file found, using default configuration
************* Module simplecaeser
C:  1, 0: Missing module docstring (missing-docstring)
W:  3, 0: Uses of a deprecated module 'string' (deprecated-module)
C:  5, 0: Invalid constant name "shift" (invalid-name)
C:  6, 0: Invalid constant name "choice" (invalid-name)
C:  7, 0: Invalid constant name "word" (invalid-name)
C:  8, 0: Invalid constant name "letters" (invalid-name)
C:  9, 0: Invalid constant name "encoded" (invalid-name)
C: 16,12: Operator not preceded by a space
            encoded=encoded + letters[x]
                   ^ (no-space-before-operator)


Report
======
19 statements analysed.

Duplication
-----------

+-------------------------+------+---------+-----------+
|                         |now   |previous |difference |
+=========================+======+=========+===========+
|nb duplicated lines      |0     |0        |=          |
+-------------------------+------+---------+-----------+
|percent duplicated lines |0.000 |0.000    |=          |
+-------------------------+------+---------+-----------+



Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |21     |87.50 |21       |=          |
+----------+-------+------+---------+-----------+
```

(continues on next page)

(continued from previous page)

```
|docstring |0      |0.00  |0        |=          |
+----------+-------+------+---------+-----------+
|comment   |1      |4.17  |1        |=          |
+----------+-------+------+---------+-----------+
|empty     |2      |8.33  |2        |=          |
+----------+-------+------+---------+-----------+



Statistics by type
------------------

+---------+-------+-----------+-----------+------------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+===========+===========+============+=========+
|module   |1      |1          |=          |0.00        |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|class    |0      |0          |=          |0.00        |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|method   |0      |0          |=          |0.00        |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|function |0      |0          |=          |0.00        |0.00     |
+---------+-------+-----------+-----------+------------+---------+



Messages by category
--------------------

+-----------+-------+---------+-----------+
|type       |number |previous |difference |
+===========+=======+=========+===========+
|convention |7      |7        |=          |
+-----------+-------+---------+-----------+
|refactor   |0      |0        |=          |
+-----------+-------+---------+-----------+
|warning    |1      |1        |=          |
+-----------+-------+---------+-----------+
|error      |0      |0        |=          |
+-----------+-------+---------+-----------+



Messages
--------

+-------------------------+------------+
|message id               |occurrences |
+=========================+============+
|invalid-name             |5           |
+-------------------------+------------+
|no-space-before-operator |1           |
+-------------------------+------------+
|missing-docstring        |1           |
+-------------------------+------------+
|deprecated-module        |1           |
+-------------------------+------------+
```

(continues on next page)

(continued from previous page)

```
Global evaluation
-----------------
Your code has been rated at 5.79/10
```

Wow. That's a lot of stuff. The first part is the 'messages' section while the second part is the 'report' section. There are two points I want to tackle here.

First point is that all the tables of statistics (i.e. the report) are a bit overwhelming so I want to silence them. To do that, I will use the **–reports=n** option.

Tip: Many of Pylint's commonly used command line options have shortcuts. for example, **–reports=n** can be abbreviated to **-rn**. Pylint's man page lists all these shortcuts.

Second, previous experience taught me that the default output for the messages needed a bit more info. We can see the first line is:

```
"C:  1: Missing docstring (missing-docstring)"
```

This basically means that line 1 violates a convention `C`. It's telling me I really should have a docstring. I agree, but what if I didn't fully understand what rule I violated. Knowing only that I violated a convention isn't much help if I'm a newbie. Another information there is the message symbol between parens, `missing-docstring` here.

If I want to read up a bit more about that, I can go back to the command line and try this:

```
robertk01 Desktop$ pylint --help-msg=missing-docstring
No config file found, using default configuration
:missing-docstring (C0111): *Missing docstring*
  Used when a module, function, class or method has no docstring. Some special
  methods like __init__ doesn't necessarily require a docstring. This message
  belongs to the basic checker.
```

Yeah, ok. That one was a bit of a no-brainer but I have run into error messages that left me with no clue about what went wrong, simply because I was unfamiliar with the underlying mechanism of code theory. One error that puzzled my newbie mind was:

```
:too-many-instance-attributes (R0902): *Too many instance attributes (%s/%s)*
```

I get it now thanks to Pylint pointing it out to me. If you don't get that one, pour a fresh cup of coffee and look into it - let your programmer mind grow!

2.4 The Next Step

Now that we got some configuration stuff out of the way, let's see what we can do with the remaining warnings.

If we add a docstring to describe what the code is meant to do that will help. I'm also going to be a bit cowboy and ignore the `deprecated-module` message because I like to take risks in life. A deprecation warning means that future versions of Python may not support that code so my code may break in the future. There are 5 `invalid-name` messages that we will get to later. Lastly, I violated the convention of using spaces around an operator such as "=" so I'll fix that too. To sum up, I'll add a docstring to line 2, put spaces around the = sign on line 16 and use the `--disable=deprecated-module` to ignore the deprecation warning.

Here is the updated code:

```
 1  #!/usr/bin/env python
 2  """This script prompts a user to enter a message to encode or decode
 3  using a classic Caeser shift substitution (3 letter shift)"""
 4
 5  import string
 6
 7  shift = 3
 8  choice = raw_input("would you like to encode or decode?")
 9  word = (raw_input("Please enter text"))
10  letters = string.ascii_letters + string.punctuation + string.digits
11  encoded = ''
12  if choice == "encode":
13      for letter in word:
14          if letter == ' ':
15              encoded = encoded + ' '
16          else:
17              x = letters.index(letter) + shift
18              encoded = encoded + letters[x]
19  if choice == "decode":
20      for letter in word:
21          if letter == ' ':
22              encoded = encoded + ' '
23          else:
24              x = letters.index(letter) - shift
25              encoded = encoded + letters[x]
26
27  print encoded
```

And here is what happens when we run it with our `--disable=deprecated-module` option:

```
robertk01 Desktop$ pylint --reports=n --disable=deprecated-module simplecaeser.py
No config file found, using default configuration
************* Module simplecaeser
C:  7, 0: Invalid constant name "shift" (invalid-name)
C:  8, 0: Invalid constant name "choice" (invalid-name)
C:  9, 0: Invalid constant name "word" (invalid-name)
C: 10, 0: Invalid constant name "letters" (invalid-name)
C: 11, 0: Invalid constant name "encoded" (invalid-name)
```

Nice! We're down to just the `invalid-name` messages.

There are fairly well defined conventions around naming things like instance variables, functions, classes, etc. The conventions focus on the use of UPPERCASE and lowercase as well as the characters that separate multiple words in the name. This lends itself well to checking via a regular expression, thus the **should match (([A-Z_][A-Z1-9_]*)|(__.*__))$**.

In this case Pylint is telling me that those variables appear to be constants and should be all UPPERCASE. This is an in-house convention that lives with Pylint since its inception. You too can create your own in-house naming conventions but for the purpose of this tutorial, we want to stick to the PEP-8 standard. In this case, the variables I declared should follow the convention of all lowercase. The appropriate rule would be something like: "should match [a-z_][a-z0-9_]{2,30}$". Notice the lowercase letters in the regular expression (a-z versus A-Z).

If we run that rule using a `--const-rgx='[a-z\_][a-z0-9\_]{2,30}$'` option, it will now be quite quiet:

```
robertk01 Desktop$ pylint --reports=n --disable=deprecated-module --const-rgx='[a-z_
↪][a-z0-9_]{2,30}$'  simplecaeser.py
No config file found, using default configuration
```

Regular expressions can be quite a beast so take my word on this particular example but go ahead and read up on them if you want.

Tip: It would really be a pain to have to use all these options on the command line all the time. That's what the configuration file is for. We can configure our Pylint to store our options for us so we don't have to declare them on the command line. Using the configuration file is a nice way of formalizing your rules and quickly sharing them with others. Invoking `pylint --generate-rcfile` will create a sample rcfile with all the options set and explained in comments.

That's it for the basic intro. More tutorials will follow.

CHAPTER 3

User Guide

3.1 Installation

3.1.1 Python packages

Pylint should be easily installable using pip.

```
python -m pip install pylint
```

3.1.2 Source distribution installation

From the source distribution, extract the tarball, go to the extracted directory and simply run

```
python setup.py install
```

Or you can install it in editable mode, using

```
python setup.py develop
```

3.2 Running Pylint

3.2.1 Invoking Pylint

Pylint is meant to be called from the command line. The usage is

```
pylint [options] modules_or_packages
```

You should give Pylint the name of a python package or module, or some number of packages or modules. Pylint `will not import` this package or module, though uses Python internals to locate them and as such is subject

to the same rules and configuration. You should pay attention to your PYTHONPATH, since it is a common error to analyze an installed version of a module instead of the development version.

It is also possible to analyze python files, with a few restrictions. The thing to keep in mind is that Pylint will try to convert the file name to a module name, and only be able to process the file if it succeeds.

```
pylint mymodule.py
```

should always work since the current working directory is automatically added on top of the python path

```
pylint directory/mymodule.py
```

will work if directory is a python package (i.e. has an __init__.py file or it is an implicit namespace package) or if "directory" is in the python path.

For more details on this see the *Frequently Asked Questions*.

It is also possible to call Pylint from an other python program, thanks to py_run() function in epylint module, assuming Pylint options are stored in pylint_options string, as:

```
from pylint import epylint as lint
lint.py_run(pylint_options)
```

To silently run Pylint on a module_name.py module, and get its standard output and error:

```
from pylint import epylint as lint
(pylint_stdout, pylint_stderr) = lint.py_run('module_name.py', return_std=True)
```

3.2.2 Command line options

First of all, we have two basic (but useful) options.

--version show program's version number and exit

-h, --help show help about the command line options

Pylint is architectured around several checkers. you can disable a specific checker or some of its messages or messages categories by specifying --disable=<symbol>. If you want to enable only some checkers or some message symbols, first use --disable=all then --enable=<symbol> with <symbol> being a comma separated list of checker names and message symbols. See the list of available features for a description of provided checkers with their functionalities. The --disable and --enable options can be used with comma separated lists mixing checkers, message ids and categories like -d C,W,no-error,design

It is possible to disable all messages with --disable=all. This is useful to enable only a few checkers or a few messages by first disabling everything, and then re-enabling only what you need.

Each checker has some specific options, which can take either a yes/no value, an integer, a python regular expression, or a comma separated list of values (which are generally used to override a regular expression in special cases). For a full list of options, use --help

Specifying all the options suitable for your setup and coding standards can be tedious, so it is possible to use a configuration file to specify the default values. You can specify a configuration file on the command line using the --rcfile option. Otherwise, Pylint searches for a configuration file in the following order and uses the first one it finds:

1. pylintrc in the current working directory
2. .pylintrc in the current working directory

3. If the current working directory is in a Python module, Pylint searches up the hierarchy of Python modules until it finds a `pylintrc` file. This allows you to specify coding standards on a module-by-module basis. Of course, a directory is judged to be a Python module if it contains an `__init__.py` file.
4. The file named by environment variable `PYLINTRC`
5. if you have a home directory which isn't `/root`:
    1. `.pylintrc` in your home directory
    2. `.config/pylintrc` in your home directory
6. `/etc/pylintrc`

The `--generate-rcfile` option will generate a commented configuration file on standard output according to the current configuration and exit. This includes:

- Any configuration file found as explained above
- Options appearing before `--generate-rcfile` on the Pylint command line

Of course you can also start with the default values and hand tune the configuration.

Other useful global options include:

--ignore=<file[,file...]> Add files or directories to the blacklist. They should be base names, not paths.

--output-format=<format> Select output format (text, json, custom).

--msg-template=<template> Modify text output message template.

--list-msgs Generate pylint's messages.

--full-documentation Generate pylint's full documentation, in reST format.

3.2.3 Parallel execution

It is possible to speed up the execution of Pylint. If the running computer has more CPUs than one, then the files for checking could be spread on all cores via Pylints's sub-processes. This functionality is exposed via `-j` command line parameter. If the provided number is 0, then the total number of CPUs will be used.

Example:

```
pylint -j 4 mymodule1.py mymodule2.py mymodule3.py mymodule4.py
```

This will spawn 4 parallel Pylint sub-process, where each provided module will be checked in parallel. Discovered problems by checkers are not displayed immediately. They are shown just after completing checking a module.

There are some limitations in running checks in parallel in current implementation. It is not possible to use custom plugins (i.e. `--load-plugins` option), nor it is not possible to use initialization hooks (i.e. `--init-hook` option).

This will spawn 4 parallel Pylint subprocesses, each provided module being checked by one or another subprocess.

3.2.4 Exit codes

Pylint returns bit-encoded exit codes. If applicable the table lists related stderr stream message output.

exit code	meaning	stderr stream message
0	no error	
1	fatal message issued	
2	error message issued	
4	warning message issued	
8	refactor message issued	
16	convention message issued	
32	usage error	• "internal error while receiving resultsfrom child linter" "Error occurred, stopping the linter." • "<return of linter.help()>" • "Jobs number <#> should be greater than 0"

3.3 Pylint output

The default format for the output is raw text. You can change this by passing pylint the `--output-format=<value>` option. Possible values are: json, parseable, colorized and msvs (visual studio).

Moreover you can customize the exact way information are displayed using the *–msg-template=<format string>* option. The *format string* uses the Python new format syntax and the following fields are available :

path relative path to the file

abspath absolute path to the file

line line number

column column number

module module name

obj object within the module (if any)

msg text of the message

msg_id the message code (eg. I0011)

symbol symbolic name of the message (eg. locally-disabled)

C one letter indication of the message category

category fullname of the message category

For example, the former (pre 1.0) default format can be obtained with:

```
pylint --msg-template='{msg_id}:{line:3d},{column}: {obj}: {msg}'
```

A few other examples:

- the new default format:

```
{C}:{line:3d},{column:2d}: {msg} ({symbol})
```

- Visual Studio compatible format (former 'msvs' output format):

```
{path}({line}): [{msg_id}{obj}] {msg}
```

- Parseable (Emacs and all, former 'parseable' output format) format:

```
{path}:{line}: [{msg_id}({symbol}), {obj}] {msg}
```

3.3.1 Source code analysis section

For each python module, Pylint will first display a few '*' characters followed by the name of the module. Then, a number of messages with the following format:

```
MESSAGE_TYPE: LINE_NUM:[OBJECT:] MESSAGE
```

You can get another output format, useful since it's recognized by most editors or other development tools using the `--output-format=parseable` option.

The message type can be:

- [R]efactor for a "good practice" metric violation
- [C]onvention for coding standard violation
- [W]arning for stylistic problems, or minor programming issues
- [E]rror for important programming issues (i.e. most probably bug)
- [F]atal for errors which prevented further processing

Sometimes the line of code which caused the error is displayed with a caret pointing to the error. This may be generalized in future versions of Pylint.

Example (extracted from a run of Pylint on itself...):

```
************* Module pylint.checkers.format
W:  50: Too long line (86/80)
W:108: Operator not followed by a space
     print >>sys.stderr, 'Unable to match %r', line
             ^
W:141: Too long line (81/80)
W:  74:searchall: Unreachable code
W:171:FormatChecker.process_tokens: Redefining built-in (type)
W:150:FormatChecker.process_tokens: Too many local variables (20/15)
W:150:FormatChecker.process_tokens: Too many branches (13/12)
```

3.3.2 Reports section

Following the analysis message, Pylint can display a set of reports, each one focusing on a particular aspect of the project, such as number of messages by categories, modules dependencies. These features can be enabled through the `--reports=y` option, or its shorthand version `-rn`.

For instance, the metrics report displays summaries gathered from the current run.

- the number of processed modules
- for each module, the percentage of errors and warnings
- the total number of errors and warnings
- percentage of classes, functions and modules with docstrings, and a comparison from the previous run

- percentage of classes, functions and modules with correct name (according to the coding standard), and a comparison from the previous run
- a list of external dependencies found in the code, and where they appear

3.3.3 Score section

Finally, Pylint displays a global evaluation score for the code, rated out of a maximum score of 10.0. This output can be suppressed through the `--score=n` option, or its shorthand version `-sn`.

The evaluation formula can be overridden with the `--evaluation=<python_expression>` option.

3.4 Messages control

An example available from the examples directory:

```
"""pylint option block-disable"""

__revision__ = None

class Foo(object):
    """block-disable test"""

    def __init__(self):
        pass

    def meth1(self, arg):
        """this issues a message"""
        print self

    def meth2(self, arg):
        """and this one not"""
        # pylint: disable=unused-argument
        print self\
              + "foo"

    def meth3(self):
        """test one line disabling"""
        # no error
        print self.bla # pylint: disable=no-member
        # error
        print self.blop

    def meth4(self):
        """test re-enabling"""
        # pylint: disable=no-member
        # no error
        print self.bla
        print self.blop
        # pylint: enable=no-member
        # error
        print self.blip

    def meth5(self):
        """test IF sub-block re-enabling"""
```

(continues on next page)

(continued from previous page)

```
        # pylint: disable=no-member
        # no error
        print self.bla
        if self.blop:
            # pylint: enable=no-member
            # error
            print self.blip
        else:
            # no error
            print self.blip
        # no error
        print self.blip

    def meth6(self):
        """test TRY/EXCEPT sub-block re-enabling"""
        # pylint: disable=no-member
        # no error
        print self.bla
        try:
            # pylint: enable=no-member
            # error
            print self.blip
        except UndefinedName: # pylint: disable=undefined-variable
            # no error
            print self.blip
        # no error
        print self.blip

    def meth7(self):
        """test one line block opening disabling"""
        if self.blop: # pylint: disable=no-member
            # error
            print self.blip
        else:
            # error
            print self.blip
        # error
        print self.blip


    def meth8(self):
        """test late disabling"""
        # error
        print self.blip
        # pylint: disable=no-member
        # no error
        print self.bla
        print self.blop
```

3.5 Configuration

3.5.1 Naming Styles

Introduction

Pylint recognizes a number of different name types internally. With a few exceptions, the type of the name is governed by the location the assignment to a name is found in, and not the type of object assigned.

Name Type	Description
`module`	Module and package names, same as the file names.
`const`	Module-level constants, any variable defined at module level that is not bound to a class object.
`class`	Names in `class` statements, as well as names bound to class objects at module level.
`function`	Functions, toplevel or nested in functions or methods.
`method`	Methods, functions defined in class bodies. Includes static and class methods.
`attr`	Attributes created on class instances inside methods.
`argument`	Arguments to any function type, including lambdas.
`variable`	Local variables in function scopes.
`class-attribute`	Attributes defined in class bodies.
`inlinevar`	Loop variables in list comprehensions and generator expressions.

Default behavior

By default, Pylint will enforce PEP8-suggested names.

Predefined Naming Styles

Pylint provides set of predefined naming styles. Those predefined naming styles may be used to adjust Pylint configuration to coding style used in linted project.

Following predefined naming styles are available:

- `snake_case`
- `camelCase`
- `PascalCase`
- `UPPER_CASE`
- `any` - fake style which does not enforce any limitations

Following options are exposed:

--module-naming-style=<style>

--const-naming-style=<style>

--class-naming-style=<style>

--function-naming-style=<style>

--method-naming-style=<style>

--attr-naming-style=<style>

--argument-naming-style=<style>

--variable-naming-style=<style>

--class-attribute-naming-style=<style>

--inlinevar-naming-style=<style>

Custom regular expressions

If predefined naming styles are too limited, checker behavior may be further customized. For each name type, a separate regular expression matching valid names of this type can be defined. If any of custom regular expressions are defined, it overrides `*-naming-style` option value.

Regular expressions for the names are anchored at the beginning, any anchor for the end must be supplied explicitly. Any name not matching the regular expression will lead to an instance of `invalid-name`.

--module-rgx=<regex>

--const-rgx=<regex>

--class-rgx=<regex>

--function-rgx=<regex>

--method-rgx=<regex>

--attr-rgx=<regex>

--argument-rgx=<regex>

--variable-rgx=<regex>

--class-attribute-rgx=<regex>

--inlinevar-rgx=<regex>

Multiple naming styles for custom regular expressions

Large code bases that have been worked on for multiple years often exhibit an evolution in style as well. In some cases, modules can be in the same package, but still have different naming style based on the stratum they belong to. However, intra-module consistency should still be required, to make changes inside a single file easier. For this case, Pylint supports regular expression with several named capturing group.

Rather than emitting name warnings immediately, Pylint will determine the prevalent naming style inside each module and enforce it on all names.

Consider the following (simplified) example:

```
pylint --function-rgx='(?:(?P<snake>[a-z_]+)|(?P<camel>_?[A-Z]+))$' sample.py
```

The regular expression defines two naming styles, `snake` for snake-case names, and `camel` for camel-case names.

In `sample.py`, the function name on line 1 and 7 will mark the module and enforce the match of named group `snake` for the remaining names in the module:

```
def valid_snake_case(arg):
    ...

def InvalidCamelCase(arg):
    ...

def more_valid_snake_case(arg):
  ...
```

Because of this, the name on line 4 will trigger an `invalid-name` warning, even though the name matches the given regex.

Matches named `exempt` or `ignore` can be used for non-tainting names, to prevent built-in or interface-dictated names to trigger certain naming styles.

--name-group=<name1:name2:...,...>
Default value: empty

Format: comma-separated groups of colon-separated names.

This option can be used to combine name styles. For example, `function:method` enforces that functions and methods use the same style, and a style triggered by either name type carries over to the other. This requires that the regular expression for the combined name types use the same group names.

Name Hints

--include-naming-hint=y|n
Default: off

Include a hint (regular expression used) for the correct name format with every `invalid-name` warning.

3.6 Editor and IDE integration

To use Pylint with:

- Emacs, see http://www.emacswiki.org/emacs/PythonProgrammingInEmacs#toc8,
- Vim, see http://www.vim.org/scripts/script.php?script_id=891,
- Visual Studio, see https://docs.microsoft.com/visualstudio/python/code-pylint,
- Eclipse and PyDev, see http://pydev.org/manual_adv_pylint.html,
- Komodo, see http://mateusz.loskot.net/posts/2006/01/15/running-pylint-from-komodo/,
- gedit, see https://launchpad.net/gedit-pylint-2 or https://wiki.gnome.org/Apps/Gedit/PylintPlugin,
- WingIDE, see http://www.wingware.com/doc/edit/pylint,
- PyCharm, see *the section below*,
- TextMate, see *the section below*

Pylint is integrated in:

- Visual Studio, see the *Python > Run PyLint* command on a project's context menu.
- Eric IDE, see the *Project > Check* menu,
- Spyder, see http://packages.python.org/spyder/pylint.html,
- pyscripter, see the *Tool -> Tools* menu.

3.6.1 Using Pylint thru flymake in Emacs

To enable flymake for Python, insert the following into your .emacs:

```
;; Configure flymake for Python
(when (load "flymake" t)
  (defun flymake-pylint-init ()
    (let* ((temp-file (flymake-init-create-temp-buffer-copy
                       'flymake-create-temp-inplace))
           (local-file (file-relative-name
                        temp-file
```

(continues on next page)

(continued from previous page)

```
                        (file-name-directory buffer-file-name))))
      (list "epylint" (list local-file))))
  (add-to-list 'flymake-allowed-file-name-masks
               '("\\.py\\'" flymake-pylint-init)))

;; Set as a minor mode for Python
(add-hook 'python-mode-hook '(lambda () (flymake-mode)))
```

Above stuff is in `pylint/elisp/pylint-flymake.el`, which should be automatically installed on Debian systems, in which cases you don't have to put it in your `.emacs` file.

Other things you may find useful to set:

```
;; Configure to wait a bit longer after edits before starting
(setq-default flymake-no-changes-timeout '3)

;; Keymaps to navigate to the errors
(add-hook 'python-mode-hook '(lambda () (define-key python-mode-map "\C-cn" 'flymake-
↪goto-next-error)))
(add-hook 'python-mode-hook '(lambda () (define-key python-mode-map "\C-cp" 'flymake-
↪goto-prev-error)))
```

Finally, by default flymake only displays the extra information about the error when you hover the mouse over the highlighted line. The following will use the minibuffer to display messages when you the cursor is on the line.

```
;; To avoid having to mouse hover for the error message, these functions make flymake␣
↪error messages
;; appear in the minibuffer
(defun show-fly-err-at-point ()
  "If the cursor is sitting on a flymake error, display the message in the minibuffer"
  (require 'cl)
  (interactive)
  (let ((line-no (line-number-at-pos)))
    (dolist (elem flymake-err-info)
      (if (eq (car elem) line-no)
      (let ((err (car (second elem))))
        (message "%s" (flymake-ler-text err)))))))

(add-hook 'post-command-hook 'show-fly-err-at-point)
```

Alternative, if you only wish to pollute the minibuffer after an explicit flymake-goto-* then use the following instead of a post-command-hook

```
(defadvice flymake-goto-next-error (after display-message activate compile)
  "Display the error in the mini-buffer rather than having to mouse over it"
  (show-fly-err-at-point))

(defadvice flymake-goto-prev-error (after display-message activate compile)
  "Display the error in the mini-buffer rather than having to mouse over it"
  (show-fly-err-at-point))
```

3.6.2 Integrate Pylint with PyCharm

Install Pylint the usual way:

```
pip install pylint
```

Remember the path at which it's installed:

```
which pylint
```

Within PyCharm:

1. Navigate to the preferences window
2. Select "External Tools"
3. Click the plus sign at the bottom of the dialog to add a new external task
4. In the dialog, populate the following fields:

   Name Pylint

   Description A Python source code analyzer which looks for programming errors, helps enforcing a coding standard and sniffs for some code smells.

   Synchronize files after execution unchecked

   Program `/path/to/pylint`

   Parameters `$FilePath$`

5. Click OK

The option to check the current file with Pylint should now be available in *Tools* > *External Tools* > *Pylint*.

3.6.3 Integrate Pylint with TextMate

Install Pylint in the usual way:

```
pip install pylint
```

Install the Python bundle for TextMate:

1. select *TextMate* > *Preferences*
2. select the *Bundles* tab
3. find and tick the *Python* bundle in the list

You should now see it in *Bundles* > *Python*.

In *Preferences*, select the *Variables* tab. If a `TM_PYCHECKER` variable is not already listed, add it, with the value `pylint`.

The default keyboard shortcut to run the syntax checker is *Control-Shift-V* - open a `.py` file in Textmate, and try it.

You should see the output in a new window:

> PyCheckMate 1.2 – Pylint 1.4.4
>
> No config file found, using default configuration

Then all is well, and most likely Pylint will have expressed some opinions about your Python code (or will exit with `0` if your code already conforms to its expectations).

If you receive a message:

> Please install PyChecker, PyFlakes, Pylint, PEP 8 or flake8 for more extensive code checking.

That means that Pylint wasn't found, which is likely an issue with command paths - TextMate needs be looking for Pylint on the right paths.

Check where Pylint has been installed, using `which`:

```
$ which pylint
/usr/local/bin/pylint
```

The output will tell you where Pylint can be found; in this case, in `/usr/local/bin`.

1. select *TextMate* > *Preferences*
2. select the *Variables* tab
3. find and check that a `PATH` variable exists, and that it contains the appropriate path (if the path to Pylint were `/usr/local/bin/pylint` as above, then the variable would need to contain `/usr/local/bin`). An actual example in this case might be `$PATH:/opt/local/bin:/usr/local/bin:/usr/texbin`, which includes other paths.

. . . and try running Pylint again.

CHAPTER 4

How To Guides

4.1 How to Write a Checker

You can find some simple examples in the distribution (custom.py and custom_raw.py).

There are three kinds of checkers:

- Raw checkers, which analyse each module as a raw file stream.
- Token checkers, which analyse a file using the list of tokens that represent the source code in the file.
- AST checkers, which work on an AST representation of the module.

The AST representation is provided by the `astroid` library. `astroid` adds additional information and methods over `ast` in the standard library, to make tree navigation and code introspection easier.

4.1.1 Writing an AST Checker

Let's implement a checker to make sure that all `return` nodes in a function return a unique constant. Firstly we will need to fill in some required boilerplate:

```
import astroid

from pylint.checkers import BaseChecker
from pylint.interfaces import IAstroidChecker

class UniqueReturnChecker(BaseChecker):
    __implements__ = IAstroidChecker

    name = 'unique-returns'
    priority = -1
    msgs = {
        'W0001': (
            'Returns a non-unique constant.',
```

(continues on next page)

(continued from previous page)

```
            'non-unique-returns',
            'All constants returned in a function should be unique.'
        ),
    }
    options = (
        (
            'ignore-ints',
            {
                'default': False, 'type': 'yn', 'metavar' : '<y_or_n>',
                'help': 'Allow returning non-unique integers',
            }
        ),
    )
```

So far we have defined the following required components of our checker:

- **A name. The name is used to generate a special configuration** section for the checker, when options have been provided.
- **A priority. This must be to be lower than 0. The checkers are ordered by** the priority when run, from the most negative to the most positive.
- **A message dictionary. Each checker is being used for finding problems** in your code, the problems being displayed to the user through **messages**. The message dictionary should specify what messages the checker is going to emit. It has the following format:

  ```
  msgs = {
      'message-id': (
          'displayed-message', 'message-symbol', 'message-help'
      )
  }
  ```

  - The `message-id` should be a 5-digit number, prefixed with a **message category**. There are multiple message categories, these being `C`, `W`, `E`, `F`, `R`, standing for `Convention`, `Warning`, `Error`, `Fatal` and `Refactoring`. The rest of the 5 digits should not conflict with existing checkers and they should be consistent across the checker. For instance, the first two digits should not be different across the checker.
  - The `displayed-message` is used for displaying the message to the user, once it is emitted.
  - The `message-symbol` is an alias of the message id and it can be used wherever the message id can be used.
  - The `message-help` is used when calling `pylint --help-msg`.

We have also defined an optional component of the checker. The options list defines any user configurable options. It has the following format:

```
options = (
    'option-symbol': {'argparse-like-kwarg': 'value'},
)
```

- The `option-symbol` is a unique name for the option. This is used on the command line and in config files. The hyphen is replaced by an underscore when used in the checker, similarly to how you would use `argparse.Namespace`.

Next we'll track when we enter and leave a function.

```
def __init__(self, linter=None):
    super(UniqueReturnChecker, self).__init__(linter)
    self._function_stack = []

def visit_functiondef(self, node):
    self._function_stack.append([])

def leave_functiondef(self, node):
    self._function_stack.pop()
```

In the constructor we initialise a stack to keep a list of return nodes for each function. An AST checker is a visitor, and should implement `visit_<lowered class name>` or `leave_<lowered class name>` methods for the nodes it's interested in. In this case we have implemented `visit_functiondef` and `leave_functiondef` to add a new list of return nodes for this function, and to remove the list of return nodes when we leave the function.

Finally we'll implement the check. We will define a `visit_return` function, which is called with a `astroid.node_classes.Return` node.

We'll need to be able to figure out what attributes a `astroid.node_classes.Return` node has available. We can use `astroid.extract_node()` for this:

```
>>> node = astroid.extract_node("return 5")
>>> node
<Return l.1 at 0x7efe62196390>
>>> help(node)
>>> node.value
<Const.int l.1 at 0x7efe62196ef0>
```

We could also construct a more complete example:

```
>>> node_a, node_b = astroid.extract_node("""
... def test():
...     if True:
...         return 5 #@
...     return 5 #@
""")
>>> node_a.value
<Const.int l.4 at 0x7efe621a74e0>
>>> node_a.value.value
5
>>> node_a.value.value == node_b.value.value
True
```

For `astroid.extract_node()`, you can use `#@` at the end of a line to choose which statements will be extracted into nodes.

For more information on `astroid.extract_node()`, see the astroid documentation.

Now we know how to use the astroid node, we can implement our check.

```
def visit_return(self, node):
    if not isinstance(node.value, astroid.node_classes.Const):
        return

    for other_return in self._function_stack[-1]:
       if (node.value.value == other_return.value.value and
           not (self.config.ignore_ints and node.value.pytype() == int)):
           self.add_message(
```

(continues on next page)

(continued from previous page)

```
                'non-unique-returns', node=node,
            )

        self._function_stack[-1].append(node)
```

Once we have established that the source code has failed our check, we use `add_message()` to emit our failure message.

Finally, we need to register the checker with pylint. Add the `register` function to the top level of the file.

```
def register(linter):
    linter.register_checker(UniqueReturnChecker(linter))
```

We are now ready to debug and test our checker!

4.1.2 Debugging a Checker

It is very simple to get to a point where we can use `pdb`. We'll need a small test case. Put the following into a Python file:

```
def test():
    if True:
        return 5
    return 5

def test2():
    if True:
        return 1
    return 5
```

After inserting pdb into our checker and installing it, we can run pylint with only our checker:

```
$ pylint --load-plugins=my_plugin --disable=all --enable=non-unique-returns test.py
(Pdb)
```

Now we can debug our checker!

Note: `my_plugin` refers to a module called `my_plugin.py`. This module can be made available to pylint by putting this module's parent directory in your `PYTHONPATH` environment variable or by adding the `my_plugin.py` file to the `pylint/checkers` directory if running from source.

4.1.3 Testing a Checker

Pylint is very well suited to test driven development. You can implement the template of the checker, produce all of your test cases and check that they fail, implement the checker, then check that all of your test cases work.

Pylint provides a `pylint.testutils.CheckerTestCase` to make test cases very simple. We can use the example code that we used for debugging as our test cases.

```
import my_plugin
import pylint.testutils

```

(continues on next page)

(continued from previous page)

```
class TestUniqueReturnChecker(pylint.testutils.CheckerTestCase):
    CHECKER_CLASS = my_plugin.UniqueReturnChecker

    def test_finds_non_unique_ints(self):
        func_node, return_node_a, return_node_b = astroid.extract_node("""
        def test(): #@
            if True:
                return 5 #@
            return 5 #@
        """)

        self.checker.visit_functiondef(func_node)
        self.checker.visit_return(return_node_a)
        with self.assertAddsMessages(
            pylint.testutils.Message(
                msg_id='non-unique-returns',
                node=return_node_b,
            ),
        ):
            self.checker.visit_return(return_node_b)

    def test_ignores_unique_ints(self):
        func_node, return_node_a, return_node_b = astroid.extract_node("""
        def test(): #@
            if True:
                return 1 #@
            return 5 #@
        """)

        with self.assertNoMessages():
            self.checker.visit_functiondef(func_node)
            self.checker.visit_return(return_node_a)
            self.checker.visit_return(return_node_b)
```

Once again we are using `astroid.extract_node()` to construct our test cases. `pylint.testutils.CheckerTestCase` has created the linter and checker for us, we simply simulate a traversal of the AST tree using the nodes that we are interested in.

4.2 How To Write a Pylint Plugin

Pylint provides support for writing two types of extensions. First, there is the concept of **checkers**, which can be used for finding problems in your code. Secondly, there is also the concept of **transform plugin**, which represents a way through which the inference and the capabilities of Pylint can be enhanced and tailored to a particular module, library of framework.

In general, a plugin is a module which should have a function `register`, which takes an instance of `pylint.lint.PyLinter` as input.

So a basic hello-world plugin can be implemented as:

```
# Inside hello_plugin.py
def register(linter):
  print 'Hello world'
```

We can run this plugin by placing this module in the PYTHONPATH and invoking **pylint** as:

```
$ pylint -E --load-plugins hello_plugin foo.py
Hello world
```

Depending if we need a **transform plugin** or a **checker**, this might not be enough. For the former, this is enough to declare the module as a plugin, but in the case of the latter, we need to register our checker with the linter object, by calling the following inside the `register` function:

```
linter.register_checker(OurChecker(linter))
```

For more information on writing a checker see *How to Write a Checker*.

4.3 Transform plugins

4.3.1 Why write a plugin?

Pylint is a static analysis tool and Python is a dynamically typed language. So there will be cases where Pylint cannot analyze files properly (this problem can happen in statically typed languages also if reflection or dynamic evaluation is used).

The plugins are a way to tell Pylint how to handle such cases, since only the user would know what needs to be done. They are usually operating on the AST level, by modifying or changing it in a way which can ease its understanding by Pylint.

4.3.2 Example

Let us run Pylint on a module from the Python source: warnings.py and see what happens:

```
amitdev$ pylint -E Lib/warnings.py
E:297,36: Instance of 'WarningMessage' has no 'message' member (no-member)
E:298,36: Instance of 'WarningMessage' has no 'filename' member (no-member)
E:298,51: Instance of 'WarningMessage' has no 'lineno' member (no-member)
E:298,64: Instance of 'WarningMessage' has no 'line' member (no-member)
```

Did we catch a genuine error? Let's open the code and look at `WarningMessage` class:

```
class WarningMessage(object):

  """Holds the result of a single showwarning() call."""

  _WARNING_DETAILS = ("message", "category", "filename", "lineno", "file",
                      "line")

  def __init__(self, message, category, filename, lineno, file=None,
                   line=None):
    local_values = locals()
    for attr in self._WARNING_DETAILS:
      setattr(self, attr, local_values[attr])
    self._category_name = category.__name__ if category else None

  def __str__(self):
    ...
```

Ah, the fields (`message`, `category` etc) are not defined statically on the class. Instead they are added using `setattr`. Pylint would have a tough time figuring this out.

4.3.3 Enter Plugin

We can write a transform plugin to tell Pylint how to analyze this properly.

One way to fix our example with a plugin would be to transform the `WarningMessage` class, by setting the attributes so that Pylint can see them. This can be done by registering a transform function. We can transform any node in the parsed AST like Module, Class, Function etc. In our case we need to transform a class. It can be done so:

```
import astroid
from astroid import MANAGER

def register(linter):
  # Needed for registering the plugin.
  pass

def transform(cls):
  if cls.name == 'WarningMessage':
    import warnings
    for f in warnings.WarningMessage._WARNING_DETAILS:
      cls.locals[f] = [astroid.Class(f, None)]

MANAGER.register_transform(astroid.Class, transform)
```

Let's go through the plugin. First, we need to register a class transform, which is done via the `register_transform` function in `MANAGER`. It takes the node type and function as parameters. We need to change a class, so we use `astroid.Class`. We also pass a `transform` function which does the actual transformation.

`transform` function is simple as well. If the class is `WarningMessage` then we add the attributes to its locals (we are not bothered about type of attributes, so setting them as class will do. But we could set them to any type we want). That's it.

Note: We don't need to do anything in the `register` function of the plugin since we are not modifying anything in the linter itself.

Lets run Pylint with this plugin and see:

```
amitdev$ pylint -E --load-plugins warning_plugin Lib/warnings.py
amitdev$
```

All the false positives associated with `WarningMessage` are now gone. This is just an example, any code transformation can be done by plugins.

See astroid/brain for real life examples of transform plugins.

CHAPTER 5

Technical Reference

5.1 Startup and the Linter Class

The two main classes in `pylint.lint` are `pylint.lint.Run` and `pylint.lint.PyLinter`.

The `pylint.lint.Run` object is responsible for starting up pylint. It does some basic checking of the given command line options to find the initial hook to run, find the config file to use, and find which plugins have been specified. It can then create the master `pylint.lint.PyLinter` instance and initialise it with the config file and plugins that were discovered when preprocessing the command line options. Finally the `pylint.lint.Run` object launches any child linters for parallel jobs, and starts the linting process.

The `pylint.lint.PyLinter` is responsible for coordinating the linting process. It parses the configuration and provides it for the checkers and other plugins, it handles the messages emitted by the checkers, it handles the output reporting, and it launches the checkers.

5.2 Checkers

All of the default pylint checkers exist in `pylint.checkers`. This is where most of pylint's brains exist. Most checkers are AST based and so use `astroid`. `pylint.checkers.utils` provides a large number of utility methods for dealing with `astroid`.

5.3 Optional Pylint checkers in the extensions module

Pylint provides the following optional plugins:

- *Deprecated Builtins checker*
- *Else If Used checker*
- *Compare-To-Zero checker*
- *Parameter Documentation checker*

- *Docstyle checker*
- *Compare-To-Empty-String checker*
- *Design checker*
- *Overlap-Except checker*
- *Multiple Types checker*

You can activate any or all of these extensions by adding a `load-plugins` line to the `MASTER` section of your `.pylintrc`, for example:

```
load-plugins=pylint.extensions.docparams,pylint.extensions.docstyle
```

5.3.1 Deprecated Builtins checker

This checker is provided by `pylint.extensions.bad_builtin`. Verbatim name of the checker is `deprecated_builtins`.

Documentation

This used to be the `bad-builtin` core checker, but it was moved to an extension instead. It can be used for finding prohibited used builtins, such as `map` or `filter`, for which other alternatives exists.

If you want to control for what builtins the checker should warn about, you can use the `bad-functions` option:

```
$ pylint a.py --load-plugins=pylint.extensions.bad_builtin --bad-functions=apply,
↪reduce
...
```

Options

bad-functions List of builtins function names that should not be used, separated by a comma

Default: `map,filter`

Messages

bad-builtin (W0141) *Used builtin function %s* Used when a black listed builtin function is used (see the bad-function option). Usual black listed functions are the ones like map, or filter , where Python offers now some cleaner alternative like list comprehension.

5.3.2 Else If Used checker

This checker is provided by `pylint.extensions.check_elif`. Verbatim name of the checker is `else_if_used`.

Messages

else-if-used (R5501) *Consider using "elif" instead of "else if"* Used when an else statement is immediately followed by an if statement and does not contain statements that would be unrelated to it.

5.3.3 Compare-To-Zero checker

This checker is provided by `pylint.extensions.comparetozero`. Verbatim name of the checker is `compare-to-zero`.

Messages

compare-to-zero (C2001) *Avoid comparisons to zero* Used when Pylint detects comparison to a 0 constant.

5.3.4 Parameter Documentation checker

This checker is provided by `pylint.extensions.docparams`. Verbatim name of the checker is `parameter_documentation`.

Documentation

If you document the parameters of your functions, methods and constructors and their types systematically in your code this optional component might be useful for you. Sphinx style, Google style, and Numpy style are supported. (For some examples, see https://pypi.python.org/pypi/sphinxcontrib-napoleon .)

You can activate this checker by adding the line:

```
load-plugins=pylint.extensions.docparams
```

to the `MASTER` section of your `.pylintrc`.

This checker verifies that all function, method, and constructor docstrings include documentation of the

- parameters and their types
- return value and its type
- exceptions raised

and can handle docstrings in

- Sphinx style (`param`, `type`, `return`, `rtype`, `raise` / `except`):

```
def function_foo(x, y, z):
    '''function foo ...

    :param x: bla x
    :type x: int

    :param y: bla y
    :type y: float

    :param int z: bla z

    :return: sum
    :rtype: float

    :raises OSError: bla
    '''
    return x + y + z
```

- or the Google style (`Args:`, `Returns:`, `Raises:`):

```
def function_foo(x, y, z):
    '''function foo ...

    Args:
        x (int): bla x
        y (float): bla y

        z (int): bla z

    Returns:
        float: sum

    Raises:
        OSError: bla
    '''
    return x + y + z
```

- or the Numpy style (`Parameters`, `Returns`, `Raises`):

```
def function_foo(x, y, z):
    '''function foo ...

    Parameters
    ----------
    x: int
        bla x
    y: float
        bla y

    z: int
        bla z

    Returns
    -------
    float
        sum

    Raises
    ------
    OSError
        bla
    '''
    return x + y + z
```

You'll be notified of **missing parameter documentation** but also of **naming inconsistencies** between the signature and the documentation which often arise when parameters are renamed automatically in the code, but not in the documentation.

Constructor parameters can be documented in either the class docstring or the `__init__` docstring, but not both:

```
class ClassFoo(object):
    '''Sphinx style docstring foo

    :param float x: bla x

    :param y: bla y
```

(continues on next page)

(continued from previous page)

```
    :type y: int
    '''
    def __init__(self, x, y):
        pass

class ClassBar(object):
    def __init__(self, x, y):
        '''Google style docstring bar

        Args:
            x (float): bla x
            y (int): bla y
        '''
        pass
```

In some cases, having to document all parameters is a nuisance, for instance if many of your functions or methods just follow a **common interface**. To remove this burden, the checker accepts missing parameter documentation if one of the following phrases is found in the docstring:

- For the other parameters, see
- For the parameters, see

(with arbitrary whitespace between the words). Please add a link to the docstring defining the interface, e.g. a superclass method, after "see":

```
def callback(x, y, z):
    '''Sphinx style docstring for callback ...

    :param x: bla x
    :type x: int

    For the other parameters, see
    :class:`MyFrameworkUsingAndDefiningCallback`
    '''
    return x + y + z

def callback(x, y, z):
    '''Google style docstring for callback ...

    Args:
        x (int): bla x

    For the other parameters, see
    :class:`MyFrameworkUsingAndDefiningCallback`
    '''
    return x + y + z
```

Naming inconsistencies in existing parameter and their type documentations are still detected.

Options

accept-no-param-doc Whether to accept totally missing parameter documentation in the docstring of a function that has parameters.

Default: `yes`

accept-no-raise-doc Whether to accept totally missing raises documentation in the docstring of a function that raises an exception.

Default: `yes`

accept-no-return-doc Whether to accept totally missing return documentation in the docstring of a function that returns a statement.

Default: `yes`

accept-no-yields-doc Whether to accept totally missing yields documentation in the docstring of a generator.

Default: `yes`

default-docstring-type If the docstring type cannot be guessed the specified docstring type will be used.

Default: `default`

Messages

differing-param-doc (W9017) *"%s" differing in parameter documentation* Please check parameter names in declarations.

differing-type-doc (W9018) *"%s" differing in parameter type documentation* Please check parameter names in type declarations.

multiple-constructor-doc (W9005) *"%s" has constructor parameters documented in class and __init__* Please remove parameter declarations in the class or constructor.

missing-param-doc (W9015) *"%s" missing in parameter documentation* Please add parameter declarations for all parameters.

missing-type-doc (W9016) *"%s" missing in parameter type documentation* Please add parameter type declarations for all parameters.

missing-raises-doc (W9006) *"%s" not documented as being raised* Please document exceptions for all raised exception types.

missing-return-doc (W9011) *Missing return documentation* Please add documentation about what this method returns.

missing-return-type-doc (W9012) *Missing return type documentation* Please document the type returned by this method.

missing-yield-doc (W9013) *Missing yield documentation* Please add documentation about what this generator yields.

missing-yield-type-doc (W9014) *Missing yield type documentation* Please document the type yielded by this method.

redundant-returns-doc (W9008) *Redundant returns documentation* Please remove the return/rtype documentation from this method.

redundant-yields-doc (W9010) *Redundant yields documentation* Please remove the yields documentation from this method.

5.3.5 Docstyle checker

This checker is provided by `pylint.extensions.docstyle`. Verbatim name of the checker is `docstyle`.

Messages

bad-docstring-quotes (C0198) *Bad docstring quotes in %s, expected """, given %s* Used when a docstring does not have triple double quotes.

docstring-first-line-empty (C0199) *First line empty in %s docstring* Used when a blank line is found at the beginning of a docstring.

5.3.6 Compare-To-Empty-String checker

This checker is provided by `pylint.extensions.emptystring`. Verbatim name of the checker is `compare-to-empty-string`.

Messages

compare-to-empty-string (C1901) *Avoid comparisons to empty string* Used when Pylint detects comparison to an empty string constant.

5.3.7 Design checker

This checker is provided by `pylint.extensions.mccabe`. Verbatim name of the checker is `design`.

Documentation

You can now use this plugin for finding complexity issues in your code base.

Activate it through `pylint --load-plugins=pylint.extensions.mccabe`. It introduces a new warning, `too-complex`, which is emitted when a code block has a complexity higher than a preestablished value, which can be controlled through the `max-complexity` option, such as in this example:

```
$ cat a.py
def f10():
    """McCabe rating: 11"""
    myint = 2
    if myint == 5:
        return myint
    elif myint == 6:
        return myint
    elif myint == 7:
        return myint
    elif myint == 8:
        return myint
    elif myint == 9:
        return myint
    elif myint == 10:
        if myint == 8:
            while True:
                return True
        elif myint == 8:
            with myint:
                return 8
    else:
        if myint == 2:
```

(continues on next page)

(continued from previous page)

```
            return myint
        return myint
    return myint
$ pylint a.py --load-plugins=pylint.extensions.mccabe
R:1: 'f10' is too complex. The McCabe rating is 11 (too-complex)
$ pylint a.py --load-plugins=pylint.extensions.mccabe --max-complexity=50
$
```

Options

max-complexity McCabe complexity cyclomatic threshold

Default: `10`

Messages

too-complex (R1260) *%s is too complex. The McCabe rating is %d* Used when a method or function is too complex based on McCabe Complexity Cyclomatic

5.3.8 Overlap-Except checker

This checker is provided by `pylint.extensions.overlapping_exceptions`. Verbatim name of the checker is `overlap-except`.

Messages

overlapping-except (W0714) *Overlapping exceptions (%s)* Used when exceptions in handler overlap or are identical

5.3.9 Multiple Types checker

This checker is provided by `pylint.extensions.redefined_variable_type`. Verbatim name of the checker is `multiple_types`.

Messages

redefined-variable-type (R0204) *Redefinition of %s type from %s to %s* Used when the type of a variable changes inside a method or a function.

5.4 Pylint features

5.4.1 Pylint global options and switches

Pylint provides global options and switches.

General options

ignore Add files or directories to the blacklist. They should be base names, not paths.

Default: `CVS`

ignore-patterns Add files or directories matching the regex patterns to the blacklist. The regex matches against base names, not paths.

persistent Pickle collected data for later comparisons.

Default: `yes`

load-plugins List of plugins (as comma separated values of python modules names) to load, usually to register additional checkers.

jobs Use multiple processes to speed up Pylint.

Default: `1`

unsafe-load-any-extension Allow loading of arbitrary C extensions. Extensions are imported into the active Python interpreter and may run arbitrary code.

extension-pkg-whitelist A comma-separated list of package or module names from where C extensions may be loaded. Extensions are loading into the active Python interpreter and may run arbitrary code

suggestion-mode When enabled, pylint would attempt to guess common misconfiguration and emit user-friendly hints instead of false-positive error messages

Default: `yes`

exit-zero Always return a 0 (non-error) status code, even if lint errors are found. This is primarily useful in continuous integration scripts.

Messages control options

confidence Only show warnings with the listed confidence levels. Leave empty to show all. Valid levels: HIGH, INFERENCE, INFERENCE_FAILURE, UNDEFINED

enable Enable the message, report, category or checker with the given id(s). You can either give multiple identifier separated by comma (,) or put this option multiple time (only on the command line, not in the configuration file where it should appear only once). See also the "–disable" option for examples.

disable Disable the message, report, category or checker with the given id(s). You can either give multiple identifiers separated by comma (,) or put this option multiple times (only on the command line, not in the configuration file where it should appear only once).You can also use "–disable=all" to disable everything first and then reenable specific checks. For example, if you want to run only the similarities checker, you can use "–disable=all –enable=similarities". If you want to run only the classes checker, but have no Warning level messages displayed, use"–disable=all –enable=classes –disable=W"

Default: `print-statement,parameter-unpacking,unpacking-in-except,
old-raise-syntax,backtick,long-suffix,old-ne-operator,
old-octal-literal,import-star-module-level,non-ascii-bytes-literal,
invalid-unicode-literal,apply-builtin,basestring-builtin,
buffer-builtin,cmp-builtin,coerce-builtin,execfile-builtin,
file-builtin,long-builtin,raw_input-builtin,reduce-builtin,
standarderror-builtin,unicode-builtin,xrange-builtin,
coerce-method,delslice-method,getslice-method,setslice-method,
no-absolute-import,old-division,dict-iter-method,dict-view-method,
next-method-called,metaclass-assignment,indexing-exception,`

```
raising-string,reload-builtin,oct-method,hex-method,
nonzero-method,cmp-method,input-builtin,round-builtin,
intern-builtin,unichr-builtin,map-builtin-not-iterating,
zip-builtin-not-iterating,range-builtin-not-iterating,
filter-builtin-not-iterating,using-cmp-argument,eq-without-hash,
div-method,idiv-method,rdiv-method,exception-message-attribute,
invalid-str-codec,sys-max-int,bad-python3-import,
deprecated-string-function,deprecated-str-translate-call,
deprecated-itertools-function,deprecated-types-field,
next-method-defined,dict-items-not-iterating,
dict-keys-not-iterating,dict-values-not-iterating,
deprecated-operator-function,deprecated-urllib-function,
xreadlines-attribute,deprecated-sys-function,exception-escape,
comprehension-escape
```

Reports options

output-format Set the output format. Available formats are text, parseable, colorized, json and msvs (visual studio).You can also give a reporter class, eg mypackage.mymodule.MyReporterClass.

Default: `text`

reports Tells whether to display a full report or only the messages

evaluation Python expression which should return a note less than 10 (10 is the highest note). You have access to the variables errors warning, statement which respectively contain the number of errors / warnings messages and the total number of statements analyzed. This is used by the global evaluation report (RP0004).

Default: `10.0 - ((float(5 * error + warning + refactor + convention) / statement) * 10)`

score Activate the evaluation score.

Default: `yes`

msg-template Template used to display messages. This is a python new-style format string used to format the message information. See doc for all details

5.4.2 Pylint checkers' options and switches

Pylint checkers can provide three set of features:

- options that control their execution,
- messages that they can raise,
- reports that they can generate.

Below is a list of all checkers and their features.

Async checker

Verbatim name of the checker is `async`.

Messages

not-async-context-manager (E1701) *Async context manager '%s' doesn't implement __aenter__ and __aexit__.* Used when an async context manager is used with an object that does not implement the async context management protocol. This message can't be emitted when using Python < 3.5.

yield-inside-async-function (E1700) *Yield inside async function* Used when an *yield* or *yield from* statement is found inside an async function. This message can't be emitted when using Python < 3.5.

Basic checker

Verbatim name of the checker is `basic`.

Options

good-names Good variable names which should always be accepted, separated by a comma

Default: `i,j,k,ex,Run,_`

bad-names Bad variable names which should always be refused, separated by a comma

Default: `foo,bar,baz,toto,tutu,tata`

name-group Colon-delimited sets of names that determine each other's naming style when the name regexes allow several styles.

include-naming-hint Include a hint for the correct naming format with invalid-name

property-classes List of decorators that produce properties, such as abc.abstractproperty. Add to this list to register other decorators that produce valid properties.

Default: `abc.abstractproperty`

class-attribute-naming-style Naming style matching correct class attribute names

Default: `any`

class-attribute-rgx Regular expression matching correct class attribute names. Overrides class-attribute-naming-style

variable-naming-style Naming style matching correct variable names

Default: `snake_case`

variable-rgx Regular expression matching correct variable names. Overrides variable- naming-style

argument-naming-style Naming style matching correct argument names

Default: `snake_case`

argument-rgx Regular expression matching correct argument names. Overrides argument- naming-style

method-naming-style Naming style matching correct method names

Default: `snake_case`

method-rgx Regular expression matching correct method names. Overrides method-naming- style

inlinevar-naming-style Naming style matching correct inline iteration names

Default: `any`

inlinevar-rgx Regular expression matching correct inline iteration names. Overrides inlinevar-naming-style

class-naming-style Naming style matching correct class names

Default: `PascalCase`

class-rgx Regular expression matching correct class names. Overrides class-naming-style

function-naming-style Naming style matching correct function names

Default: `snake_case`

function-rgx Regular expression matching correct function names. Overrides function- naming-style

attr-naming-style Naming style matching correct attribute names

Default: `snake_case`

attr-rgx Regular expression matching correct attribute names. Overrides attr-naming- style

module-naming-style Naming style matching correct module names

Default: `snake_case`

module-rgx Regular expression matching correct module names. Overrides module-naming- style

const-naming-style Naming style matching correct constant names

Default: `UPPER_CASE`

const-rgx Regular expression matching correct constant names. Overrides const-naming- style

no-docstring-rgx Regular expression which should only match function or class names that do not require a docstring.

Default: `^_`

docstring-min-length Minimum line length for functions/classes that require docstrings, shorter ones are exempt.

Default: `-1`

Messages

not-in-loop (E0103) *%r not properly in loop* Used when break or continue keywords are used outside a loop.

function-redefined (E0102) *%s already defined line %s* Used when a function / class / method is redefined.

continue-in-finally (E0116) *'continue' not supported inside 'finally' clause* Emitted when the *continue* keyword is found inside a finally clause, which is a SyntaxError.

abstract-class-instantiated (E0110) *Abstract class %r with abstract methods instantiated* Used when an abstract class with *abc.ABCMeta* as metaclass has abstract methods and is instantiated.

star-needs-assignment-target (E0114) *Can use starred expression only in assignment target* Emitted when a star expression is not used in an assignment target. This message can't be emitted when using Python < 3.0.

duplicate-argument-name (E0108) *Duplicate argument name %s in function definition* Duplicate argument names in function definitions are syntax errors.

return-in-init (E0101) *Explicit return in __init__* Used when the special class method __init__ has an explicit return value.

too-many-star-expressions (E0112) *More than one starred expression in assignment* Emitted when there are more than one starred expressions (*x) in an assignment. This is a SyntaxError. This message can't be emitted when using Python < 3.0.

nonlocal-and-global (E0115) *Name %r is nonlocal and global* Emitted when a name is both nonlocal and global. This message can't be emitted when using Python < 3.0.

used-prior-global-declaration (E0118) *Name %r is used prior to global declaration* Emitted when a name is used prior a global declaration, which results in an error since Python 3.6. This message can't be emitted when using Python < 3.6.

return-outside-function (E0104) *Return outside function* Used when a "return" statement is found outside a function or method.

return-arg-in-generator (E0106) *Return with argument inside generator* Used when a "return" statement with an argument is found outside in a generator function or method (e.g. with some "yield" statements). This message can't be emitted when using Python >= 3.3.

invalid-star-assignment-target (E0113) *Starred assignment target must be in a list or tuple* Emitted when a star expression is used as a starred assignment target. This message can't be emitted when using Python < 3.0.

bad-reversed-sequence (E0111) *The first reversed() argument is not a sequence* Used when the first argument to reversed() builtin isn't a sequence (does not implement __reversed__, nor __getitem__ and __len__

nonexistent-operator (E0107) *Use of the non-existent %s operator* Used when you attempt to use the C-style pre-increment orpre-decrement operator – and ++, which doesn't exist in Python.

yield-outside-function (E0105) *Yield outside function* Used when a "yield" statement is found outside a function or method.

init-is-generator (E0100) *__init__ method is a generator* Used when the special class method __init__ is turned into a generator by a yield in its body.

nonlocal-without-binding (E0117) *nonlocal name %s found without binding* Emitted when a nonlocal variable does not have an attached name somewhere in the parent scopes This message can't be emitted when using Python < 3.0.

lost-exception (W0150) *%s statement in finally block may swallow exception* Used when a break or a return statement is found inside the finally clause of a try...finally block: the exceptions raised in the try clause will be silently swallowed instead of being re-raised.

assert-on-tuple (W0199) *Assert called on a 2-uple. Did you mean 'assert x,y'?* A call of assert on a tuple will always evaluate to true if the tuple is not empty, and will always evaluate to false if it is.

dangerous-default-value (W0102) *Dangerous default value %s as argument* Used when a mutable value as list or dictionary is detected in a default value for an argument.

duplicate-key (W0109) *Duplicate key %r in dictionary* Used when a dictionary expression binds the same key multiple times.

useless-else-on-loop (W0120) *Else clause on loop without a break statement* Loops should only have an else clause if they can exit early with a break statement, otherwise the statements under else should be on the same scope as the loop itself.

expression-not-assigned (W0106) *Expression "%s" is assigned to nothing* Used when an expression that is not a function call is assigned to nothing. Probably something else was intended.

confusing-with-statement (W0124) *Following "as" with another context manager looks like a tuple.* Emitted when a *with* statement component returns multiple values and uses name binding with *as* only for a part of those values, as in with ctx() as a, b. This can be misleading, since it's not clear if the context manager returns a tuple or if the node without a name binding is another context manager.

unnecessary-lambda (W0108) *Lambda may not be necessary* Used when the body of a lambda expression is a function call on the same argument list as the lambda itself; such lambda expressions are in all but a few cases replaceable with the function being called in the body of the lambda.

assign-to-new-keyword (W0111) *Name %s will become a keyword in Python %s* Used when assignment will become invalid in future Python release due to introducing new keyword

pointless-statement (W0104) *Statement seems to have no effect* Used when a statement doesn't have (or at least seems to) any effect.

pointless-string-statement (W0105) *String statement has no effect* Used when a string is used as a statement (which of course has no effect). This is a particular case of W0104 with its own message so you can easily disable it if you're using those strings as documentation, instead of comments.

unnecessary-pass (W0107) *Unnecessary pass statement* Used when a "pass" statement that can be avoided is encountered.

unreachable (W0101) *Unreachable code* Used when there is some code behind a "return" or "raise" statement, which will never be accessed.

eval-used (W0123) *Use of eval* Used when you use the "eval" function, to discourage its usage. Consider using *ast.literal_eval* for safely evaluating strings containing Python expressions from untrusted sources.

exec-used (W0122) *Use of exec* Used when you use the "exec" statement (function for Python 3), to discourage its usage. That doesn't mean you cannot use it !

using-constant-test (W0125) *Using a conditional statement with a constant value* Emitted when a conditional statement (If or ternary if) uses a constant value for its test. This might not be what the user intended to do.

deprecated-lambda (W0110) *map/filter on lambda could be replaced by comprehension* Used when a lambda is the first argument to "map" or "filter". It could be clearer as a list comprehension or generator expression. This message can't be emitted when using Python >= 3.0.

literal-comparison (R0123) *Comparison to literal* Used when comparing an object to a literal, which is usually what you do not want to do, since you can compare to a different literal than what was expected altogether.

invalid-name (C0103) *%s name "%s" doesn't conform to %s* Used when the name doesn't conform to naming rules associated to its type (constant, variable, class...).

blacklisted-name (C0102) *Black listed name "%s"* Used when the name is listed in the black list (unauthorized names).

misplaced-comparison-constant (C0122) *Comparison should be %s* Used when the constant is placed on the left side of a comparison. It is usually clearer in intent to place it in the right hand side of the comparison.

singleton-comparison (C0121) *Comparison to %s should be %s* Used when an expression is compared to singleton values like True, False or None.

unneeded-not (C0113) *Consider changing "%s" to "%s"* Used when a boolean expression contains an unneeded negation.

empty-docstring (C0112) *Empty %s docstring* Used when a module, function, class or method has an empty docstring (it would be too easy ;).

missing-docstring (C0111) *Missing %s docstring* Used when a module, function, class or method has no docstring.Some special methods like __init__ doesn't necessary require a docstring.

unidiomatic-typecheck (C0123) *Using type() instead of isinstance() for a typecheck.* The idiomatic way to perform an explicit typecheck in Python is to use isinstance(x, Y) rather than type(x) == Y, type(x) is Y. Though there are unusual situations where these give different results.

Reports

RP0101 Statistics by type

Classes checker

Verbatim name of the checker is `classes`.

Options

defining-attr-methods List of method names used to declare (i.e. assign) instance attributes.

Default: `__init__,__new__,setUp`

valid-classmethod-first-arg List of valid names for the first argument in a class method.

Default: `cls`

valid-metaclass-classmethod-first-arg List of valid names for the first argument in a metaclass class method.

Default: `mcs`

exclude-protected List of member names, which should be excluded from the protected access warning.

Default: `_asdict,_fields,_replace,_source,_make`

Messages

access-member-before-definition (E0203) *Access to member %r before its definition line %s* Used when an instance member is accessed before it's actually assigned.

method-hidden (E0202) *An attribute defined in %s line %s hides this method* Used when a class defines a method which is hidden by an instance attribute from an ancestor class or set by some client code.

assigning-non-slot (E0237) *Assigning to attribute %r not defined in class slots* Used when assigning to an attribute not defined in the class slots.

duplicate-bases (E0241) *Duplicate bases for class %r* Used when a class has duplicate bases.

inconsistent-mro (E0240) *Inconsistent method resolution order for class %r* Used when a class has an inconsistent method resolution order.

inherit-non-class (E0239) *Inheriting %r, which is not a class.* Used when a class inherits from something which is not a class.

invalid-slots (E0238) *Invalid __slots__ object* Used when an invalid __slots__ is found in class. Only a string, an iterable or a sequence is permitted.

invalid-slots-object (E0236) *Invalid object %r in __slots__, must contain only non empty strings* Used when an invalid (non-string) object occurs in __slots__.

no-method-argument (E0211) *Method has no argument* Used when a method which should have the bound instance as first argument has no argument defined.

no-self-argument (E0213) *Method should have "self" as first argument* Used when a method has an attribute different the "self" as first argument. This is considered as an error since this is a so common convention that you shouldn't break it!

unexpected-special-method-signature (E0302) *The special method %r expects %s param(s), %d %s given* Emitted when a special method was defined with an invalid number of parameters. If it has too few or too many, it might not work at all.

non-iterator-returned (E0301) *__iter__ returns non-iterator* Used when an __iter__ method returns something which is not an iterable (i.e. has no *__next__* method)

invalid-length-returned (E0303) *__len__ does not return non-negative integer* Used when an __len__ method returns something which is not a non-negative integer

protected-access (W0212) *Access to a protected member %s of a client class* Used when a protected member (i.e. class member with a name beginning with an underscore) is access outside the class or a descendant of the class where it's defined.

attribute-defined-outside-init (W0201) *Attribute %r defined outside __init__* Used when an instance attribute is defined outside the __init__ method.

no-init (W0232) *Class has no __init__ method* Used when a class has no __init__ method, neither its parent classes.

abstract-method (W0223) *Method %r is abstract in class %r but is not overridden* Used when an abstract method (i.e. raise NotImplementedError) is not overridden in concrete class.

arguments-differ (W0221) *Parameters differ from %s %r method* Used when a method has a different number of arguments than in the implemented interface or in an overridden method.

signature-differs (W0222) *Signature differs from %s %r method* Used when a method signature is different than in the implemented interface or in an overridden method.

bad-staticmethod-argument (W0211) *Static method with %r as first argument* Used when a static method has "self" or a value specified in valid- classmethod-first-arg option or valid-metaclass-classmethod-first-arg option as first argument.

useless-super-delegation (W0235) *Useless super delegation in method %r* Used whenever we can detect that an overridden method is useless, relying on super() delegation to do the same thing as another method from the MRO.

non-parent-init-called (W0233) *__init__ method from a non direct base class %r is called* Used when an __init__ method is called on a class which is not in the direct ancestors for the analysed class.

super-init-not-called (W0231) *__init__ method from base class %r is not called* Used when an ancestor class method has an __init__ method which is not called by a derived class.

no-classmethod-decorator (R0202) *Consider using a decorator instead of calling classmethod* Used when a class method is defined without using the decorator syntax.

no-staticmethod-decorator (R0203) *Consider using a decorator instead of calling staticmethod* Used when a static method is defined without using the decorator syntax.

no-self-use (R0201) *Method could be a function* Used when a method doesn't use its bound instance, and so could be written as a function.

single-string-used-for-slots (C0205) *Class __slots__ should be a non-string iterable* Used when a class __slots__ is a simple string, rather than an iterable.

bad-classmethod-argument (C0202) *Class method %s should have %s as first argument* Used when a class method has a first argument named differently than the value specified in valid-classmethod-first-arg option (default to "cls"), recommended to easily differentiate them from regular instance methods.

bad-mcs-classmethod-argument (C0204) *Metaclass class method %s should have %s as first argument* Used when a metaclass class method has a first argument named differently than the value specified in valid-metaclass-classmethod-first-arg option (default to "mcs"), recommended to easily differentiate them from regular instance methods.

bad-mcs-method-argument (C0203) *Metaclass method %s should have %s as first argument* Used when a metaclass method has a first argument named differently than the value specified in valid-classmethod-first-arg option (default to "cls"), recommended to easily differentiate them from regular instance methods.

method-check-failed (F0202) *Unable to check methods signature (%s / %s)* Used when Pylint has been unable to check methods signature compatibility for an unexpected reason. Please report this kind if you don't make sense of it.

Design checker

Verbatim name of the checker is `design`.

Options

max-args Maximum number of arguments for function / method

Default: `5`

max-locals Maximum number of locals for function / method body

Default: `15`

max-returns Maximum number of return / yield for function / method body

Default: `6`

max-branches Maximum number of branch for function / method body

Default: `12`

max-statements Maximum number of statements in function / method body

Default: `50`

max-parents Maximum number of parents for a class (see R0901).

Default: `7`

max-attributes Maximum number of attributes for a class (see R0902).

Default: `7`

min-public-methods Minimum number of public methods for a class (see R0903).

Default: `2`

max-public-methods Maximum number of public methods for a class (see R0904).

Default: `20`

max-bool-expr Maximum number of boolean expressions in a if statement

Default: `5`

Messages

too-few-public-methods (R0903) *Too few public methods (%s/%s)* Used when class has too few public methods, so be sure it's really worth it.

too-many-ancestors (R0901) *Too many ancestors (%s/%s)* Used when class has too many parent classes, try to reduce this to get a simpler (and so easier to use) class.

too-many-arguments (R0913) *Too many arguments (%s/%s)* Used when a function or method takes too many arguments.

too-many-boolean-expressions (R0916) *Too many boolean expressions in if statement (%s/%s)* Used when a if statement contains too many boolean expressions

too-many-branches (R0912) *Too many branches (%s/%s)* Used when a function or method has too many branches, making it hard to follow.

too-many-instance-attributes (R0902) *Too many instance attributes (%s/%s)* Used when class has too many instance attributes, try to reduce this to get a simpler (and so easier to use) class.

too-many-locals (R0914) *Too many local variables (%s/%s)* Used when a function or method has too many local variables.

too-many-public-methods (R0904) *Too many public methods (%s/%s)* Used when class has too many public methods, try to reduce this to get a simpler (and so easier to use) class.

too-many-return-statements (R0911) *Too many return statements (%s/%s)* Used when a function or method has too many return statement, making it hard to follow.

too-many-statements (R0915) *Too many statements (%s/%s)* Used when a function or method has too many statements. You should then split it in smaller functions / methods.

Exceptions checker

Verbatim name of the checker is `exceptions`.

Options

overgeneral-exceptions Exceptions that will emit a warning when being caught. Defaults to "Exception"

Default: `Exception`

Messages

bad-except-order (E0701) *Bad except clauses order (%s)* Used when except clauses are not in the correct order (from the more specific to the more generic). If you don't fix the order, some exceptions may not be caught by the most specific handler.

catching-non-exception (E0712) *Catching an exception which doesn't inherit from Exception: %s* Used when a class which doesn't inherit from Exception is used as an exception in an except clause.

bad-exception-context (E0703) *Exception context set to something which is not an exception, nor None* Used when using the syntax "raise ... from ...", where the exception context is not an exception, nor None. This message can't be emitted when using Python < 3.0.

notimplemented-raised (E0711) *NotImplemented raised - should raise NotImplementedError* Used when NotImplemented is raised instead of NotImplementedError

raising-bad-type (E0702) *Raising %s while only classes or instances are allowed* Used when something which is neither a class, an instance or a string is raised (i.e. a *TypeError* will be raised).

raising-non-exception (E0710) *Raising a new style class which doesn't inherit from BaseException* Used when a new style class which doesn't inherit from BaseException is raised.

misplaced-bare-raise (E0704) *The raise statement is not inside an except clause* Used when a bare raise is not used inside an except clause. This generates an error, since there are no active exceptions to be reraised. An exception to this rule is represented by a bare raise inside a finally clause, which might work, as long as an exception is raised inside the try block, but it is nevertheless a code smell that must not be relied upon.

duplicate-except (W0705) *Catching previously caught exception type %s* Used when an except catches a type that was already caught by a previous handler.

broad-except (W0703) *Catching too general exception %s* Used when an except catches a too general exception, possibly burying unrelated errors.

raising-format-tuple (W0715) *Exception arguments suggest string formatting might be intended* Used when passing multiple arguments to an exception constructor, the first of them a string literal containing what appears to be placeholders intended for formatting

nonstandard-exception (W0710) *Exception doesn't inherit from standard "Exception" class* Used when a custom exception class is raised but doesn't inherit from the builtin "Exception" class. This message can't be emitted when using Python >= 3.0.

binary-op-exception (W0711) *Exception to catch is the result of a binary "%s" operation* Used when the exception to catch is of the form "except A or B:". If intending to catch multiple, rewrite as "except (A, B):"

bare-except (W0702) *No exception type(s) specified* Used when an except clause doesn't specify exceptions type to catch.

Format checker

Verbatim name of the checker is `format`.

Options

max-line-length Maximum number of characters on a single line.

Default: `100`

ignore-long-lines Regexp for a line that is allowed to be longer than the limit.

Default: `^\s*(# )?<?https?://\S+>?$`

single-line-if-stmt Allow the body of an if to be on the same line as the test if there is no else.

single-line-class-stmt Allow the body of a class to be on the same line as the declaration if body contains single statement.

no-space-check List of optional constructs for which whitespace checking is disabled. *dict- separator* is used to allow tabulation in dicts, etc.: {1 : 1,n222: 2}. *trailing-comma* allows a space between comma and closing bracket: (a,). *empty-line* allows space-only lines.

Default: `trailing-comma,dict-separator`

max-module-lines Maximum number of lines in a module

Default: `1000`

indent-string String used as indentation unit. This is usually " " (4 spaces) or "t" (1 tab).

Default: `'    '`

indent-after-paren Number of spaces of indent required inside a hanging or continued line.

Default: `4`

expected-line-ending-format Expected format of line ending, e.g. empty (any line ending), LF or CRLF.

Messages

bad-indentation (W0311) *Bad indentation. Found %s %s, expected %s* Used when an unexpected number of indentation's tabulations or spaces has been found.

mixed-indentation (W0312) *Found indentation with %ss instead of %ss* Used when there are some mixed tabs and spaces in a module.

unnecessary-semicolon (W0301) *Unnecessary semicolon* Used when a statement is ended by a semicolon (";"), which isn't necessary (that's python, not C ;).

lowercase-l-suffix (W0332) *Use of "l" as long integer identifier* Used when a lower case "l" is used to mark a long integer. You should use a upper case "L" since the letter "l" looks too much like the digit "1" This message can't be emitted when using Python >= 3.0.

bad-whitespace (C0326) *%s space %s %s %s* Used when a wrong number of spaces is used around an operator, bracket or block opener.

missing-final-newline (C0304) *Final newline missing* Used when the last line in a file is missing a newline.

line-too-long (C0301) *Line too long (%s/%s)* Used when a line is longer than a given number of characters.

mixed-line-endings (C0327) *Mixed line endings LF and CRLF* Used when there are mixed (LF and CRLF) newline signs in a file.

multiple-statements (C0321) *More than one statement on a single line* Used when more than on statement are found on the same line.

too-many-lines (C0302) *Too many lines in module (%s/%s)* Used when a module has too much lines, reducing its readability.

trailing-newlines (C0305) *Trailing newlines* Used when there are trailing blank lines in a file.

trailing-whitespace (C0303) *Trailing whitespace* Used when there is whitespace between the end of a line and the newline.

unexpected-line-ending-format (C0328) *Unexpected line ending format. There is '%s' while it should be '%s'.* Used when there is different newline than expected.

superfluous-parens (C0325) *Unnecessary parens after %r keyword* Used when a single item in parentheses follows an if, for, or other keyword.

bad-continuation (C0330) *Wrong %s indentation%s%s.* TODO

Imports checker

Verbatim name of the checker is `imports`.

Options

deprecated-modules Deprecated modules which should not be used, separated by a comma

Default: `optparse,tkinter.tix`

import-graph Create a graph of every (i.e. internal and external) dependencies in the given file (report RP0402 must not be disabled)

ext-import-graph Create a graph of external dependencies in the given file (report RP0402 must not be disabled)

int-import-graph Create a graph of internal dependencies in the given file (report RP0402 must not be disabled)

known-standard-library Force import order to recognize a module as part of the standard compatibility libraries.

known-third-party Force import order to recognize a module as part of a third party library.

Default: `enchant`

analyse-fallback-blocks Analyse import fallback blocks. This can be used to support both Python 2 and 3 compatible code, which means that the block might have code that exists only in one or another interpreter, leading to false positives when analysed.

allow-wildcard-with-all Allow wildcard imports from modules that define __all__.

Messages

relative-beyond-top-level (E0402) *Attempted relative import beyond top-level package* Used when a relative import tries to access too many levels in the current package.

import-error (E0401) *Unable to import %s* Used when pylint has been unable to import a module.

import-self (W0406) *Module import itself* Used when a module is importing itself.

reimported (W0404) *Reimport %r (imported line %s)* Used when a module is reimported multiple times.

relative-import (W0403) *Relative import %r, should be %r* Used when an import relative to the package directory is detected. This message can't be emitted when using Python >= 3.0.

deprecated-module (W0402) *Uses of a deprecated module %r* Used a module marked as deprecated is imported.

wildcard-import (W0401) *Wildcard import %s* Used when *from module import ** is detected.

misplaced-future (W0410) *__future__ import is not the first non docstring statement* Python 2.5 and greater require __future__ import to be the first non docstring statement in the module.

cyclic-import (R0401) *Cyclic import (%s)* Used when a cyclic import between two or more modules is detected.

wrong-import-order (C0411) *%s should be placed before %s* Used when PEP8 import order is not respected (standard imports first, then third-party libraries, then local imports)

wrong-import-position (C0413) *Import "%s" should be placed at the top of the module* Used when code and imports are mixed

ungrouped-imports (C0412) *Imports from package %s are not grouped* Used when imports are not grouped by packages

multiple-imports (C0410) *Multiple imports on one line (%s)* Used when import statement importing multiple modules is detected.

Reports

RP0401 External dependencies

RP0402 Modules dependencies graph

Iterable Check checker

Verbatim name of the checker is `iterable_check`.

Messages

not-an-iterable (E1133) *Non-iterable value %s is used in an iterating context* Used when a non-iterable value is used in place where iterable is expected

not-a-mapping (E1134) *Non-mapping value %s is used in a mapping context* Used when a non-mapping value is used in place where mapping is expected

Len checker

Verbatim name of the checker is `len`.

Messages

len-as-condition (C1801) *Do not use 'len(SEQUENCE)' to determine if a sequence is empty* Used when Pylint detects that len(sequence) is being used inside a condition to determine if a sequence is empty. Instead of comparing the length to 0, rely on the fact that empty sequences are false.

Logging checker

Verbatim name of the checker is `logging`.

Options

logging-modules Logging modules to check that the string format arguments are in logging function parameter format

Default: `logging`

Messages

logging-format-truncated (E1201) *Logging format string ends in middle of conversion specifier* Used when a logging statement format string terminates before the end of a conversion specifier.

logging-too-few-args (E1206) *Not enough arguments for logging format string* Used when a logging format string is given too few arguments.

logging-too-many-args (E1205) *Too many arguments for logging format string* Used when a logging format string is given too many arguments.

logging-unsupported-format (E1200) *Unsupported logging format character %r (%#02x) at index %d* Used when an unsupported format character is used in a logging statement format string.

logging-not-lazy (W1201) *Specify string format arguments as logging function parameters* Used when a logging statement has a call form of "logging.<logging method>(format_string % (format_args...))". Such calls should leave string interpolation to the logging method itself and be written "logging.<logging method>(format_string, format_args...)" so that the program may avoid incurring the cost of the interpolation in those cases in which no message will be logged. For more, see http://www.python.org/dev/peps/pep-0282/.

logging-format-interpolation (W1202) *Use % formatting in logging functions and pass the % parameters as arguments* Used when a logging statement has a call form of "logging.<logging method>(format_string.format(format_args...))". Such calls should use % formatting instead, but leave interpolation to the logging function by passing the parameters as arguments.

Metrics checker

Verbatim name of the checker is `metrics`.

Reports

RP0701 Raw metrics

Miscellaneous checker

Verbatim name of the checker is `miscellaneous`.

Options

notes List of note tags to take in consideration, separated by a comma.

Default: `FIXME,XXX,TODO`

Messages

fixme (W0511) Used when a warning note as FIXME or XXX is detected.

invalid-encoded-data (W0512) *Cannot decode using encoding "%s", unexpected byte at position %d* Used when a source line cannot be decoded using the specified source file encoding. This message can't be emitted when using Python >= 3.0.

Newstyle checker

Verbatim name of the checker is `newstyle`.

Messages

bad-super-call (E1003) *Bad first argument %r given to super()* Used when another argument than the current class is given as first argument of the super builtin.

missing-super-argument (E1004) *Missing argument to super()* Used when the super builtin didn't receive an argument. This message can't be emitted when using Python >= 3.0.

slots-on-old-class (E1001) *Use of __slots__ on an old style class* Used when an old style class uses the __slots__ attribute. This message can't be emitted when using Python >= 3.0.

super-on-old-class (E1002) *Use of super on an old style class* Used when an old style class uses the super builtin. This message can't be emitted when using Python >= 3.0.

property-on-old-class (W1001) *Use of "property" on an old style class* Used when Pylint detect the use of the builtin "property" on an old style class while this is relying on new style classes features. This message can't be emitted when using Python >= 3.0.

old-style-class (C1001) *Old-style class defined.* Used when a class is defined that does not inherit from another class and does not inherit explicitly from "object". This message can't be emitted when using Python >= 3.0.

Python3 checker

Verbatim name of the checker is `python3`.

Messages

unpacking-in-except (E1603) *Implicit unpacking of exceptions is not supported in Python 3* Python3 will not allow implicit unpacking of exceptions in except clauses. See http://www.python.org/dev/peps/pep-3110/

import-star-module-level (E1609) *Import * only allowed at module level* Used when the import star syntax is used somewhere else than the module level. This message can't be emitted when using Python >= 3.0.

non-ascii-bytes-literal (E1610) *Non-ascii bytes literals not supported in 3.x* Used when non-ascii bytes literals are found in a program. They are no longer supported in Python 3. This message can't be emitted when using Python >= 3.0.

parameter-unpacking (E1602) *Parameter unpacking specified* Used when parameter unpacking is specified for a function(Python 3 doesn't allow it)

long-suffix (E1606) *Use of long suffix* Used when "l" or "L" is used to mark a long integer. This will not work in Python 3, since *int* and *long* types have merged. This message can't be emitted when using Python >= 3.0.

old-octal-literal (E1608) *Use of old octal literal* Used when encountering the old octal syntax, removed in Python 3. To use the new syntax, prepend 0o on the number. This message can't be emitted when using Python >= 3.0.

old-ne-operator (E1607) *Use of the <> operator* Used when the deprecated "<>" operator is used instead of "!=". This is removed in Python 3. This message can't be emitted when using Python >= 3.0.

backtick (E1605) *Use of the `` operator* Used when the deprecated "``" (backtick) operator is used instead of the str() function.

old-raise-syntax (E1604) *Use raise ErrorClass(args) instead of raise ErrorClass, args.* Used when the alternate raise syntax 'raise foo, bar' is used instead of 'raise foo(bar)'.

print-statement (E1601) *print statement used* Used when a print statement is used (*print* is a function in Python 3)

invalid-unicode-literal (E1611) *unicode raw string literals not supported in 3.x* Used when raw unicode literals are found in a program. They are no longer supported in Python 3. This message can't be emitted when using Python >= 3.0.

deprecated-types-field (W1652) *Accessing a deprecated fields on the types module* Used when accessing a field on types that has been removed in Python 3.

deprecated-itertools-function (W1651) *Accessing a deprecated function on the itertools module* Used when accessing a function on itertools that has been removed in Python 3.

deprecated-string-function (W1649) *Accessing a deprecated function on the string module* Used when accessing a string function that has been deprecated in Python 3.

deprecated-operator-function (W1657) *Accessing a removed attribute on the operator module* Used when accessing a field on operator module that has been removed in Python 3.

deprecated-sys-function (W1660) *Accessing a removed attribute on the sys module* Used when accessing a field on sys module that has been removed in Python 3.

deprecated-urllib-function (W1658) *Accessing a removed attribute on the urllib module* Used when accessing a field on urllib module that has been removed or moved in Python 3.

xreadlines-attribute (W1659) *Accessing a removed xreadlines attribute* Used when accessing the xreadlines() function on a file stream, removed in Python 3.

metaclass-assignment (W1623) *Assigning to a class's __metaclass__ attribute* Used when a metaclass is specified by assigning to __metaclass__ (Python 3 specifies the metaclass as a class statement argument)

next-method-called (W1622) *Called a next() method on an object* Used when an object's next() method is called (Python 3 uses the next() built- in function)

dict-iter-method (W1620) *Calling a dict.iter*() method* Used for calls to dict.iterkeys(), itervalues() or iteritems() (Python 3 lacks these methods)

dict-view-method (W1621) *Calling a dict.view*() method* Used for calls to dict.viewkeys(), viewvalues() or viewitems() (Python 3 lacks these methods)

exception-message-attribute (W1645) *Exception.message removed in Python 3* Used when the message attribute is accessed on an Exception. Use str(exception) instead.

eq-without-hash (W1641) *Implementing __eq__ without also implementing __hash__* Used when a class implements __eq__ but not __hash__. In Python 2, objects get object.__hash__ as the default implementation, in Python 3 objects get None as their default __hash__ implementation if they also implement __eq__.

indexing-exception (W1624) *Indexing exceptions will not work on Python 3* Indexing exceptions will not work on Python 3. Use *exception.args[index]* instead.

bad-python3-import (W1648) *Module moved in Python 3* Used when importing a module that no longer exists in Python 3.

raising-string (W1625) *Raising a string exception* Used when a string exception is raised. This will not work on Python 3.

standarderror-builtin (W1611) *StandardError built-in referenced* Used when the StandardError built-in function is referenced (missing from Python 3)

comprehension-escape (W1662) *Using a variable that was bound inside a comprehension* Emitted when using a variable, that was bound in a comprehension handler, outside of the comprehension itself. On Python 3 these variables will be deleted outside of the comprehension.

exception-escape (W1661) *Using an exception object that was bound by an except handler* Emitted when using an exception, that was bound in an except handler, outside of the except handler. On Python 3 these exceptions will be deleted once they get out of the except handler.

deprecated-str-translate-call (W1650) *Using str.translate with deprecated deletechars parameters* Used when using the deprecated deletechars parameters from str.translate. Use re.sub to remove the desired characters

using-cmp-argument (W1640) *Using the cmp argument for list.sort / sorted* Using the cmp argument for list.sort or the sorted builtin should be avoided, since it was removed in Python 3. Using either *key* or *functools.cmp_to_key* should be preferred.

cmp-method (W1630) *__cmp__ method defined* Used when a __cmp__ method is defined (method is not used by Python 3)

coerce-method (W1614) *__coerce__ method defined* Used when a __coerce__ method is defined (method is not used by Python 3)

delslice-method (W1615) *__delslice__ method defined* Used when a __delslice__ method is defined (method is not used by Python 3)

div-method (W1642) *__div__ method defined* Used when a __div__ method is defined. Using *__truediv__* and setting__div__ = __truediv__ should be preferred.(method is not used by Python 3)

getslice-method (W1616) *__getslice__ method defined* Used when a __getslice__ method is defined (method is not used by Python 3)

hex-method (W1628) *__hex__ method defined* Used when a __hex__ method is defined (method is not used by Python 3)

idiv-method (W1643) *__idiv__ method defined* Used when an __idiv__ method is defined. Using *__itruediv__* and setting__idiv__ = __itruediv__ should be preferred.(method is not used by Python 3)

nonzero-method (W1629) *__nonzero__ method defined* Used when a __nonzero__ method is defined (method is not used by Python 3)

oct-method (W1627) *__oct__ method defined* Used when an __oct__ method is defined (method is not used by Python 3)

rdiv-method (W1644) *__rdiv__ method defined* Used when a __rdiv__ method is defined. Using *__rtruediv__* and setting__rdiv__ = __rtruediv__ should be preferred.(method is not used by Python 3)

setslice-method (W1617) *__setslice__ method defined* Used when a __setslice__ method is defined (method is not used by Python 3)

apply-builtin (W1601) *apply built-in referenced* Used when the apply built-in function is referenced (missing from Python 3)

basestring-builtin (W1602) *basestring built-in referenced* Used when the basestring built-in function is referenced (missing from Python 3)

buffer-builtin (W1603) *buffer built-in referenced* Used when the buffer built-in function is referenced (missing from Python 3)

cmp-builtin (W1604) *cmp built-in referenced* Used when the cmp built-in function is referenced (missing from Python 3)

coerce-builtin (W1605) *coerce built-in referenced* Used when the coerce built-in function is referenced (missing from Python 3)

dict-items-not-iterating (W1654) *dict.items referenced when not iterating* Used when dict.items is referenced in a non-iterating context (returns an iterator in Python 3)

dict-keys-not-iterating (W1655) *dict.keys referenced when not iterating* Used when dict.keys is referenced in a non-iterating context (returns an iterator in Python 3)

dict-values-not-iterating (W1656) *dict.values referenced when not iterating* Used when dict.values is referenced in a non-iterating context (returns an iterator in Python 3)

old-division (W1619) *division w/o __future__ statement* Used for non-floor division w/o a float literal or `from __future__ import division` (Python 3 returns a float for int division unconditionally)

execfile-builtin (W1606) *execfile built-in referenced* Used when the execfile built-in function is referenced (missing from Python 3)

file-builtin (W1607) *file built-in referenced* Used when the file built-in function is referenced (missing from Python 3)

filter-builtin-not-iterating (W1639) *filter built-in referenced when not iterating* Used when the filter built-in is referenced in a non-iterating context (returns an iterator in Python 3)

no-absolute-import (W1618) *import missing 'from __future__ import absolute_import'* Used when an import is not accompanied by `from __future__ import absolute_import` (default behaviour in Python 3)

input-builtin (W1632) *input built-in referenced* Used when the input built-in is referenced (backwards-incompatible semantics in Python 3)

intern-builtin (W1634) *intern built-in referenced* Used when the intern built-in is referenced (Moved to sys.intern in Python 3)

long-builtin (W1608) *long built-in referenced* Used when the long built-in function is referenced (missing from Python 3)

map-builtin-not-iterating (W1636) *map built-in referenced when not iterating* Used when the map built-in is referenced in a non-iterating context (returns an iterator in Python 3)

next-method-defined (W1653) *next method defined* Used when a next method is defined that would be an iterator in Python 2 but is treated as a normal function in Python 3.

invalid-str-codec (W1646) *non-text encoding used in str.decode* Used when using str.encode or str.decode with a non-text encoding. Use codecs module to handle arbitrary codecs.

range-builtin-not-iterating (W1638) *range built-in referenced when not iterating* Used when the range built-in is referenced in a non-iterating context (returns an iterator in Python 3)

raw_input-builtin (W1609) *raw_input built-in referenced* Used when the raw_input built-in function is referenced (missing from Python 3)

reduce-builtin (W1610) *reduce built-in referenced* Used when the reduce built-in function is referenced (missing from Python 3)

reload-builtin (W1626) *reload built-in referenced* Used when the reload built-in function is referenced (missing from Python 3). You can use instead imp.reload or importlib.reload.

round-builtin (W1633) *round built-in referenced* Used when the round built-in is referenced (backwards-incompatible semantics in Python 3)

sys-max-int (W1647) *sys.maxint removed in Python 3* Used when accessing sys.maxint. Use sys.maxsize instead.

unichr-builtin (W1635) *unichr built-in referenced* Used when the unichr built-in is referenced (Use chr in Python 3)

unicode-builtin (W1612) *unicode built-in referenced* Used when the unicode built-in function is referenced (missing from Python 3)

xrange-builtin (W1613) *xrange built-in referenced* Used when the xrange built-in function is referenced (missing from Python 3)

zip-builtin-not-iterating (W1637) *zip built-in referenced when not iterating* Used when the zip built-in is referenced in a non-iterating context (returns an iterator in Python 3)

Refactoring checker

Verbatim name of the checker is `refactoring`.

Options

max-nested-blocks Maximum number of nested blocks for function / method body

Default: `5`

never-returning-functions Complete name of functions that never returns. When checking for inconsistent-return-statements if a never returning function is called then it will be considered as an explicit return statement and no message will be printed.

Default: `optparse.Values,sys.exit`

Messages

simplify-boolean-expression (R1709) *Boolean expression may be simplified to %s* Emitted when redundant pre-python 2.5 ternary syntax is used.

consider-merging-isinstance (R1701) *Consider merging these isinstance calls to isinstance(%s, (%s))* Used when multiple consecutive isinstance calls can be merged into one.

consider-using-ternary (R1706) *Consider using ternary (%s)* Used when one of known pre-python 2.5 ternary syntax is used.

trailing-comma-tuple (R1707) *Disallow trailing comma tuple* In Python, a tuple is actually created by the comma symbol, not by the parentheses. Unfortunately, one can actually create a tuple by misplacing a trailing comma, which can lead to potential weird bugs in your code. You should always use parentheses explicitly for creating a tuple. This message can't be emitted when using Python < 3.0.

stop-iteration-return (R1708) *Do not raise StopIteration in generator, use return statement instead* According to PEP479, the raise of StopIteration to end the loop of a generator may lead to hard to find bugs. This PEP specify that raise StopIteration has to be replaced by a simple return statement This message can't be emitted when using Python < 3.0.

inconsistent-return-statements (R1710) *Either all return statements in a function should return an expression, or none of them should.* According to PEP8, if any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable)

redefined-argument-from-local (R1704) *Redefining argument with the local name %r* Used when a local name is redefining an argument, which might suggest a potential error. This is taken in account only for a handful of name binding operations, such as for iteration, with statement assignment and exception handler assignment.

simplifiable-if-statement (R1703) *The if statement can be replaced with %s* Used when an if statement can be replaced with 'bool(test)'.

too-many-nested-blocks (R1702) *Too many nested blocks (%s/%s)* Used when a function or a method has too many nested blocks. This makes the code less understandable and maintainable.

no-else-return (R1705) *Unnecessary "else" after "return"* Used in order to highlight an unnecessary block of code following an if containing a return statement. As such, it will warn when it encounters an else following a chain of ifs, all of them containing a return statement.

consider-iterating-dictionary (C0201) *Consider iterating the dictionary directly instead of calling .keys()* Emitted when the keys of a dictionary are iterated through the .keys() method. It is enough to just iterate through the dictionary itself, as in "for key in dictionary".

consider-using-enumerate (C0200) *Consider using enumerate instead of iterating with range and len* Emitted when code that iterates with range and len is encountered. Such code can be simplified by using the enumerate builtin.

Similarities checker

Verbatim name of the checker is `similarities`.

Options

min-similarity-lines Minimum lines number of a similarity.

Default: `4`

ignore-comments Ignore comments when computing similarities.

Default: `yes`

ignore-docstrings Ignore docstrings when computing similarities.

Default: `yes`

ignore-imports Ignore imports when computing similarities.

Messages

duplicate-code (R0801) *Similar lines in %s files* Indicates that a set of similar lines has been detected among multiple file. This usually means that the code should be refactored to avoid this duplication.

Reports

RP0801 Duplication

Spelling checker

Verbatim name of the checker is `spelling`.

Options

spelling-dict Spelling dictionary name. Available dictionaries: none. To make it working install python-enchant package.

spelling-ignore-words List of comma separated words that should not be checked.

spelling-private-dict-file A path to a file that contains private dictionary; one word per line.

spelling-store-unknown-words Tells whether to store unknown words to indicated private dictionary in –spelling-private-dict-file option instead of raising a message.

max-spelling-suggestions Limits count of emitted suggestions for spelling mistakes

Default: `4`

Messages

invalid-characters-in-docstring (C0403) *Invalid characters %r in a docstring* Used when a word in docstring cannot be checked by enchant.

wrong-spelling-in-comment (C0401) *Wrong spelling of a word '%s' in a comment:* Used when a word in comment is not spelled correctly.

wrong-spelling-in-docstring (C0402) *Wrong spelling of a word '%s' in a docstring:* Used when a word in docstring is not spelled correctly.

Stdlib checker

Verbatim name of the checker is `stdlib`.

Messages

bad-open-mode (W1501) *"%s" is not a valid mode for open.* Python supports: r, w, a[, x] modes with b, +, and U (only with r) options. See http://docs.python.org/2/library/functions.html#open

redundant-unittest-assert (W1503) *Redundant use of %s with constant value %r* The first argument of assertTrue and assertFalse is a condition. If a constant is passed as parameter, that condition will be always true. In this case a warning should be emitted.

shallow-copy-environ (W1507) *Using copy.copy(os.environ). Use os.environ.copy() instead. * os.environ is not a dict object but proxy object, so shallow copy has still effects on original object. See https://bugs.python.org/issue15373 for reference.

boolean-datetime (W1502) *Using datetime.time in a boolean context.* Using datetime.time in a boolean context can hide subtle bugs when the time they represent matches midnight UTC. This behaviour was fixed in Python 3.5. See http://bugs.python.org/issue13936 for reference. This message can't be emitted when using Python >= 3.5.

deprecated-method (W1505) *Using deprecated method %s()* The method is marked as deprecated and will be removed in a future version of Python. Consider looking for an alternative in the documentation.

bad-thread-instantiation (W1506) *threading.Thread needs the target function* The warning is emitted when a threading.Thread class is instantiated without the target function being passed. By default, the first parameter is the group param, not the target param.

String checker

Verbatim name of the checker is `string`.

Messages

format-needs-mapping (E1303) *Expected mapping for format string, not %s* Used when a format string that uses named conversion specifiers is used with an argument that is not a mapping.

truncated-format-string (E1301) *Format string ends in middle of conversion specifier* Used when a format string terminates before the end of a conversion specifier.

missing-format-string-key (E1304) *Missing key %r in format string dictionary* Used when a format string that uses named conversion specifiers is used with a dictionary that doesn't contain all the keys required by the format string.

mixed-format-string (E1302) *Mixing named and unnamed conversion specifiers in format string* Used when a format string contains both named (e.g. '%(foo)d') and unnamed (e.g. '%d') conversion specifiers. This is also used when a named conversion specifier contains * for the minimum field width and/or precision.

too-few-format-args (E1306) *Not enough arguments for format string* Used when a format string that uses unnamed conversion specifiers is given too few arguments

bad-str-strip-call (E1310) *Suspicious argument in %s.%s call* The argument to a str.{l,r,}strip call contains a duplicate character,

too-many-format-args (E1305) *Too many arguments for format string* Used when a format string that uses unnamed conversion specifiers is given too many arguments.

bad-format-character (E1300) *Unsupported format character %r (%#02x) at index %d* Used when a unsupported format character is used in a format string.

format-combined-specification (W1305) *Format string contains both automatic field numbering and manual field specification* Used when a PEP 3101 format string contains both automatic field numbering (e.g. '{}') and manual field specification (e.g. '{0}'). This message can't be emitted when using Python < 2.7.

bad-format-string-key (W1300) *Format string dictionary key should be a string, not %s* Used when a format string that uses named conversion specifiers is used with a dictionary whose keys are not all strings.

bad-format-string (W1302) *Invalid format string* Used when a PEP 3101 format string is invalid. This message can't be emitted when using Python < 2.7.

missing-format-attribute (W1306) *Missing format attribute %r in format specifier %r* Used when a PEP 3101 format string uses an attribute specifier ({0.length}), but the argument passed for formatting doesn't have that attribute. This message can't be emitted when using Python < 2.7.

missing-format-argument-key (W1303) *Missing keyword argument %r for format string* Used when a PEP 3101 format string that uses named fields doesn't receive one or more required keywords. This message can't be emitted when using Python < 2.7.

unused-format-string-argument (W1304) *Unused format argument %r* Used when a PEP 3101 format string that uses named fields is used with an argument that is not required by the format string. This message can't be emitted when using Python < 2.7.

unused-format-string-key (W1301) *Unused key %r in format string dictionary* Used when a format string that uses named conversion specifiers is used with a dictionary that contains keys not required by the format string.

invalid-format-index (W1307) *Using invalid lookup key %r in format specifier %r* Used when a PEP 3101 format string uses a lookup specifier ({a[1]}), but the argument passed for formatting doesn't contain or doesn't have that key as an attribute. This message can't be emitted when using Python < 2.7.

String Constant checker

Verbatim name of the checker is `string_constant`.

Messages

anomalous-unicode-escape-in-string (W1402) *Anomalous Unicode escape in byte string: '%s'. String constant might be missing an r or u prefix.* Used when an escape like u is encountered in a byte string where it has no effect.

anomalous-backslash-in-string (W1401) *Anomalous backslash in string: '%s'. String constant might be missing an r prefix.* Used when a backslash is in a literal string but not as an escape.

Typecheck checker

Verbatim name of the checker is `typecheck`.

Options

ignore-on-opaque-inference This flag controls whether pylint should warn about no-member and similar checks whenever an opaque object is returned when inferring. The inference can return multiple potential results while evaluating a Python object, but some branches might not be evaluated, which results in partial inference. In that case, it might be useful to still emit no-member and other checks for the rest of the inferred objects.

Default: `yes`

ignore-mixin-members Tells whether missing members accessed in mixin class should be ignored. A mixin class is detected if its name ends with "mixin" (case insensitive).

Default: `yes`

ignored-modules List of module names for which member attributes should not be checked (useful for modules/projects where namespaces are manipulated during runtime and thus existing member attributes cannot be deduced by static analysis. It supports qualified module names, as well as Unix pattern matching.

ignored-classes List of class names for which member attributes should not be checked (useful for classes with dynamically set attributes). This supports the use of qualified names.

Default: `optparse.Values,thread._local,_thread._local`

generated-members List of members which are set dynamically and missed by pylint inference system, and so shouldn't trigger E1101 when accessed. Python regular expressions are accepted.

contextmanager-decorators List of decorators that produce context managers, such as contextlib.contextmanager. Add to this list to register other decorators that produce valid context managers.

Default: `contextlib.contextmanager`

missing-member-hint-distance The minimum edit distance a name should have in order to be considered a similar match for a missing member name.

Default: `1`

missing-member-max-choices The total number of similar names that should be taken in consideration when showing a hint for a missing member.

Default: `1`

missing-member-hint Show a hint with possible names when a member name was not found. The aspect of finding the hint is based on edit distance.

Default: `yes`

Messages

unsupported-assignment-operation (E1137) *%r does not support item assignment* Emitted when an object does not support item assignment (i.e. doesn't define __setitem__ method)

unsupported-delete-operation (E1138) *%r does not support item deletion* Emitted when an object does not support item deletion (i.e. doesn't define __delitem__ method)

invalid-unary-operand-type (E1130) Emitted when a unary operand is used on an object which does not support this type of operation

unsupported-binary-operation (E1131) Emitted when a binary arithmetic operation between two operands is not supported.

no-member (E1101) *%s %r has no %r member%s* Used when a variable is accessed for an unexistent member.

not-callable (E1102) *%s is not callable* Used when an object being called has been inferred to a non callable object

redundant-keyword-arg (E1124) *Argument %r passed by position and keyword in %s call* Used when a function call would result in assigning multiple values to a function parameter, one value from a positional argument and one from a keyword argument.

assignment-from-no-return (E1111) *Assigning to function call which doesn't return* Used when an assignment is done on a function call but the inferred function doesn't return anything.

assignment-from-none (E1128) *Assigning to function call which only returns None* Used when an assignment is done on a function call but the inferred function returns nothing but None.

not-context-manager (E1129) *Context manager '%s' doesn't implement __enter__ and __exit__.* Used when an instance in a with statement doesn't implement the context manager protocol(__enter__/__exit__).

repeated-keyword (E1132) *Got multiple values for keyword argument %r in function call* Emitted when a function call got multiple values for a keyword.

invalid-metaclass (E1139) *Invalid metaclass %r used* Emitted whenever we can detect that a class is using, as a metaclass, something which might be invalid for using as a metaclass.

missing-kwoa (E1125) *Missing mandatory keyword argument %r in %s call* Used when a function call does not pass a mandatory keyword-only argument. This message can't be emitted when using Python < 3.0.

no-value-for-parameter (E1120) *No value for argument %s in %s call* Used when a function call passes too few arguments.

invalid-sequence-index (E1126) *Sequence index is not an int, slice, or instance with __index__* Used when a sequence type is indexed with an invalid type. Valid types are ints, slices, and objects with an __index__ method.

invalid-slice-index (E1127) *Slice index is not an int, None, or instance with __index__* Used when a slice index is not an integer, None, or an object with an __index__ method.

too-many-function-args (E1121) *Too many positional arguments for %s call* Used when a function call passes too many positional arguments.

unexpected-keyword-arg (E1123) *Unexpected keyword argument %r in %s call* Used when a function call passes a keyword argument that doesn't correspond to one of the function's parameter names.

unsupported-membership-test (E1135) *Value '%s' doesn't support membership test* Emitted when an instance in membership test expression doesn't implement membership protocol (__contains__/__iter__/__getitem__)

unsubscriptable-object (E1136) *Value '%s' is unsubscriptable* Emitted when a subscripted value doesn't support subscription(i.e. doesn't define __getitem__ method)

keyword-arg-before-vararg (W1113) *Keyword argument before variable positional arguments list in the definition of %s function* When defining a keyword argument before variable positional arguments, one can end up in having multiple values passed for the aforementioned parameter in case the method is called with keyword arguments.

c-extension-no-member (I1101) *%s %r has not %r member%s, but source is unavailable. Consider adding this module to extension-pkg-whitelist if you want to perform analysis based on run-time introspection of living objects.* Used when a variable is accessed for non-existent member of C extension. Due to unavailability of source static analysis is impossible, but it may be performed by introspecting living objects in run-time.

Variables checker

Verbatim name of the checker is `variables`.

Options

init-import Tells whether we should check for unused import in __init__ files.

dummy-variables-rgx A regular expression matching the name of dummy variables (i.e. expectedly not used).

Default: `_+$|(_[a-zA-Z0-9_]*[a-zA-Z0-9]+?$)|dummy|^ignored_|^unused_`

additional-builtins List of additional names supposed to be defined in builtins. Remember that you should avoid to define new builtins when possible.

callbacks List of strings which can identify a callback function by name. A callback name must start or end with one of those strings.

Default: `cb_,_cb`

redefining-builtins-modules List of qualified module names which can have objects that can redefine builtins.

Default: `six.moves,past.builtins,future.builtins,io,builtins`

ignored-argument-names Argument names that match this expression will be ignored. Default to name with leading underscore

Default: `_.*|^ignored_|^unused_`

allow-global-unused-variables Tells whether unused global variables should be treated as a violation.

Default: `yes`

Messages

unpacking-non-sequence (E0633) *Attempting to unpack a non-sequence%s* Used when something which is not a sequence is used in an unpack assignment

invalid-all-object (E0604) *Invalid object %r in __all__, must contain only strings* Used when an invalid (non-string) object occurs in __all__.

no-name-in-module (E0611) *No name %r in module %r* Used when a name cannot be found in a module.

unbalanced-tuple-unpacking (E0632) *Possible unbalanced tuple unpacking with sequence%s: left side has %d label(s), right side has %d value(s)* Used when there is an unbalanced tuple unpacking in assignment

undefined-variable (E0602) *Undefined variable %r* Used when an undefined variable is accessed.

undefined-all-variable (E0603) *Undefined variable name %r in __all__* Used when an undefined variable name is referenced in __all__.

used-before-assignment (E0601) *Using variable %r before assignment* Used when a local variable is accessed before it's assignment.

cell-var-from-loop (W0640) *Cell variable %s defined in loop* A variable used in a closure is defined in a loop. This will result in all closures using the same value for the closed-over variable.

global-variable-undefined (W0601) *Global variable %r undefined at the module level* Used when a variable is defined through the "global" statement but the variable is not defined in the module scope.

redefined-builtin (W0622) *Redefining built-in %r* Used when a variable or function override a built-in.

redefine-in-handler (W0623) *Redefining name %r from %s in exception handler* Used when an exception handler assigns the exception to an existing name

redefined-outer-name (W0621) *Redefining name %r from outer scope (line %s)* Used when a variable's name hides a name defined in the outer scope.

unused-import (W0611) *Unused %s* Used when an imported module or variable is not used.

unused-argument (W0613) *Unused argument %r* Used when a function or method argument is not used.

unused-wildcard-import (W0614) *Unused import %s from wildcard import* Used when an imported module or variable is not used from a *'from X import *'* style import.

unused-variable (W0612) *Unused variable %r* Used when a variable is defined but not used.

global-variable-not-assigned (W0602) *Using global for %r but no assignment is done* Used when a variable is defined through the "global" statement but no assignment to this variable is done.

undefined-loop-variable (W0631) *Using possibly undefined loop variable %r* Used when an loop variable (i.e. defined by a for loop or a list comprehension or a generator expression) is used outside the loop.

global-statement (W0603) *Using the global statement* Used when you use the "global" statement to update a global variable. Pylint just try to discourage this usage. That doesn't mean you cannot use it !

global-at-module-level (W0604) *Using the global statement at the module level* Used when you use the "global" statement at the module level since it has no effect

CHAPTER 6

Development

6.1 Contributing

6.1.1 Bug reports, feedback

You think you have found a bug in Pylint? Well, this may be the case since Pylint is under heavy development.

Please take the time to check if it is already in the issue tracker at https://github.com/PyCQA/pylint

If you cannot find it in the tracker, create a new issue there or discuss your problem on the code-quality@python.org mailing list.

The code-quality mailing list is also a nice place to provide feedback about Pylint, since it is shared with other tools that aim at improving the quality of python code.

Note that if you don't find something you have expected in Pylint's issue tracker, it may be because it is an issue with one of its dependencies, namely astroid:

- https://github.com/PyCQA/astroid

6.1.2 Mailing lists

You can subscribe to this mailing list at http://mail.python.org/mailman/listinfo/code-quality

Archives are available at http://mail.python.org/pipermail/code-quality/

Archives before April 2013 are available at http://lists.logilab.org/pipermail/python-projects/

6.1.3 Repository

Pylint is developed using the git distributed version control system.

You can clone Pylint and its dependencies from

```
git clone https://github.com/PyCQA/pylint
git clone https://github.com/PyCQA/astroid
```

Got a change for Pylint? Below are a few steps you should take to make sure your patch gets accepted.

- Test your code
  - Pylint is very well tested, with a high good code coverage. It has two types of tests, usual unittests and functional tests.

    The usual unittests can be found under */test* directory and they can be used for testing almost anything Pylint related. But for the ease of testing Pylint's messages, we also have the concept of functional tests.
  - You should also run all the tests to ensure that your change isn't breaking one. You can run the tests using the tox package, as in:

    ```
    python -m tox
    python -m tox -epy27 # for Python 2.7 suite only
    python -m tox -epylint # for running Pylint over Pylint's codebase
    ```
  - To run only a specific test suite, use a pattern for the test filename (**without** the `.py` extension), as in:

    ```
    python -m tox -e py27 -- -k test_functional
    python -m tox -e py27 -- -k  \*func\*
    ```
- Add a short entry to the ChangeLog describing the change, except for internal implementation only changes. Not usually required, but for changes other than small bugs we also add a couple of sentences in the release document for that release, (*What's New* section)
- Add yourself to the *CONTRIBUTORS* file, if you are not already there.
- Write a comprehensive commit message
- Relate your change to an issue in the tracker if such an issue exists (see Closing issues via commit messages of the GitHub documentation for more

  information on this)
- Document your change, if it is a non-trivial one.
- Send a pull request from GitHub (see About pull requests for more insight about this topic)

6.1.4 Functional Tests

These are residing under '/test/functional' and they are formed of multiple components. First, each Python file is considered to be a test case and it should be accompanied by a .txt file, having the same name, with the messages that are supposed to be emitted by the given test file.

In the Python file, each line for which Pylint is supposed to emit a message has to be annotated with a comment in the form `# [message_symbol]`, as in:

```
a, b, c = 1 # [unbalanced-tuple-unpacking]
```

If multiple messages are expected on the same line, then this syntax can be used:

```
a, b, c = 1.test # [unbalanced-tuple-unpacking, no-member]
```

The syntax of the .txt file has to be this:

```
symbol:line_number:function_or_class:Expected message
```

For example, this is a valid message line:

```
abstract-class-instantiated:79:main:Abstract class 'BadClass' with abstract methods␣
↪instantiated
```

If the Python file is expected to not emit any errors, then the .txt file has to be empty. If you need special control over Pylint's flag, you can also create a .rc file, which can have sections of Pylint's configuration.

During development, it's sometimes helpful to run all functional tests in your current environment in order to have faster feedback. Run with:

```
python pylint/test/test_functional.py
```

6.1.5 Tips for Getting Started with Pylint Development

- Read the *Technical Reference*. It gives a short walkthrough of the pylint codebase and will help you identify where you will need to make changes for what you are trying to implement.
- `astroid.extract_node()` is your friend. Most checkers are AST based, so you will likely need to interact with `astroid`. A short example of how to use `astroid.extract_node()` is given *here*.
- When fixing a bug for a specific check, search the code for the warning message to find where the warning is raised, and therefore where the logic for that code exists.

A Typical Development Workflow

1. Create a virtualenv in which to work:

   ```
   $ tox
   ```

2. Write the tests. See *Functional Tests*.
3. Check that the tests fail:

   ```
   $ tox
   ```

4. Fix pylint!
5. Make sure your tests pass:

   ```
   $ tox
   ```

   It is also possible to give tox a pytest specifier to run only your test:

   ```
   $ tox pylint/test/test_functional.py::test_functional
   ```

6. Package up and submit your changes as outlined in *repository*.

6.2 Building the documentation

When you would want to build the documentation for your self, you certainly can. Here's how...

6.2.1 The bare minimum

```
$ python3 -m virtualenv venv
$ . venv/bin/activate
$ pip install Sphinx
$ git clone https://github.com/PyCQA/pylint
$ cd pylint
$ python3 setup.py install
$ cd doc
$ make html
```

CHAPTER 7

Frequently Asked Questions

7.1 1. About Pylint

7.1.1 1.1 What is Pylint?

Pylint is a static code checker, meaning it can analyse your code without actually running it. Pylint checks for errors, tries to enforce a coding standard, and tries to enforce a coding style.

7.2 2. Installation

7.2.1 2.1 How do I install Pylint?

Everything should be explained on *Installation*.

7.2.2 2.2 What kind of versioning system does Pylint use?

Pylint uses the git distributed version control system. The URL of the repository is: https://github.com/PyCQA/pylint . To get the latest version of Pylint from the repository, simply invoke

```
git clone https://github.com/PyCQA/pylint
```

7.2.3 2.3 What are Pylint's dependencies?

Pylint depends on astroid and a couple of other packages. It should be compatible with any Python version greater than 2.7.0 and it is also working on PyPy.

7.2.4 2.4 What versions of Python is Pylint supporting?

Since Pylint 1.8, we support only Python 2.7 and Python 3.4+. If code uses new Python 3.6 syntax, minimal required version is Pylint 1.7.

Using this strategy really helps in maintaining a code base compatible with both versions and from this benefits not only the maintainers, but the end users as well, because it's easier to add and test new features.

7.3 3. Running Pylint

7.3.1 3.1 Can I give pylint a file as an argument instead of a module?

Pylint expects the name of a package or module as its argument. As a convenience, you can give it a file name if it's possible to guess a module name from the file's path using the python path. Some examples :

"pylint mymodule.py" should always work since the current working directory is automatically added on top of the python path

"pylint directory/mymodule.py" will work if "directory" is a python package (i.e. has an __init__.py file), an implicit namespace package or if "directory" is in the python path.

"pylint /whatever/directory/mymodule.py" will work if either:

- "/whatever/directory" is in the python path
- your cwd is "/whatever/directory"
- "directory" is a python package and "/whatever" is in the python path
- "directory" is an implicit namespace package and is in the python path.
- "directory" is a python package and your cwd is "/whatever" and so on...

7.3.2 3.2 Where is the persistent data stored to compare between successive runs?

Analysis data are stored as a pickle file in a directory which is localized using the following rules:

- value of the PYLINTHOME environment variable if set
- **".pylint.d" subdirectory of the user's home directory if it is found** (not always findable on Windows platforms)
- ".pylint.d" directory in the current directory

7.3.3 3.3 How do I find the option name (for pylintrc) corresponding to a specific command line option?

You can always generate a sample pylintrc file with –generate-rcfile Every option present on the command line before this will be included in the rc file

For example:

```
pylint --disable=bare-except,invalid-name --class-rgx='[A-Z][a-z]+' --generate-rcfile
```

7.3.4 3.4 I'd rather not run Pylint from the command line. Can I integrate it with my editor?

Much probably. Read *Editor and IDE integration*

7.4 4. Message Control

7.4.1 4.1 Is it possible to locally disable a particular message?

Yes, this feature has been added in Pylint 0.11. This may be done by adding "#pylint: disable=some-message,another-one" at the desired block level or at the end of the desired line of code

7.4.2 4.2 Is there a way to disable a message for a particular module only?

Yes, you can disable or enable (globally disabled) messages at the module level by adding the corresponding option in a comment at the top of the file:

```
# pylint: disable=wildcard-import, method-hidden
# pylint: enable=too-many-lines
```

7.4.3 4.3 How can I tell Pylint to never check a given module?

With Pylint < 0.25, add "#pylint: disable-all" at the beginning of the module. Pylint 0.26.1 and up have renamed that directive to "#pylint: skip-file" (but the first version will be kept for backward compatibility).

In order to ease finding which modules are ignored an Information-level message *file-ignored* is emitted. With recent versions of Pylint, if you use the old syntax, an additional *deprecated-disable-all* message is emitted.

7.4.4 4.4 Do I have to remember all these numbers?

No, starting from 0.25.3, you can use symbolic names for messages:

```
# pylint: disable=fixme, line-too-long
```

7.4.5 4.5 I have a callback function where I have no control over received arguments. How do I avoid getting unused argument warnings?

Prefix (ui) the callback's name by *cb_*, as in cb_onclick(...). By doing so arguments usage won't be checked. Another solution is to use one of the names defined in the "dummy-variables" configuration variable for unused argument ("_" and "dummy" by default).

7.4.6 4.6 What is the format of the configuration file?

Pylint uses ConfigParser from the standard library to parse the configuration file. It means that if you need to disable a lot of messages, you can use tricks like:

```
# disable wildcard-import, method-hidden and too-many-lines because I do
# not want it
disable= wildcard-import,
 method-hidden,
 too-many-lines
```

7.4.7 4.7 Why are there a bunch of messages disabled by default?

pylint does have some messages disabled by default, either because they are prone to false positives or that they are opinionated enough for not being included as default messages. But most of the disabled messages are from the Python 3 porting checker, which is disabled by default. It needs special activation with the `--py3k` flag.

7.5 5. Classes and Inheritance

7.5.1 5.1 When is Pylint considering a class as an abstract class?

A class is considered as an abstract class if at least one of its methods is doing nothing but raising NotImplemented-Error.

7.5.2 5.2 How do I avoid "access to undefined member" messages in my mixin classes?

To do so you have to set the ignore-mixin-members option to "yes" (this is the default value) and to name your mixin class with a name which ends with "mixin" (whatever case).

7.6 6. Troubleshooting

7.6.1 6.1 Pylint gave my code a negative rating out of ten. That can't be right!

Even though the final rating Pylint renders is nominally out of ten, there's no lower bound on it. By default, the formula to calculate score is

```
10.0 - ((float(5 * error + warning + refactor + convention) / statement) * 10)
```

However, this option can be changed in the Pylint rc file. If having negative values really bugs you, you can set the formula to be the maximum of 0 and the above expression.

7.6.2 6.2 I think I found a bug in Pylint. What should I do?

Read *Bug reports, feedback*

7.6.3 6.3 I have a question about Pylint that isn't answered here.

Read *Mailing lists*

CHAPTER 8

Some projects using Pylint

The following projects are known to use Pylint to help develop better Python code:

- edX (https://github.com/edx)
- qutebrowser (https://github.com/The-Compiler/qutebrowser)
- Odoo (https://github.com/OCA)
- Landscape.io (https://github.com/landscapeio/)
- Codacy (https://github.com/Codacy/)
- SaltStack (https://github.com/saltstack)
- many more. . .

CHAPTER 9

What's New in Pylint

High level descriptions of the most important changes between major Pylint versions.

9.1 What's New In Pylint 1.9

Release 1.9

Date 2018-05-15

9.1.1 Summary – Release highlights

- None so far

9.1.2 New checkers

- A new Python 3 checker was added to warn about the removed `operator.div` function.
- A new Python 3 checker was added to warn about accessing functions that have been moved from the urllib module in corresponding subpackages, such as `urllib.request`.

```
from urllib import urlencode
```

Instead the previous code should use `urllib.parse` or `six.moves` to import a module in a Python 2 and 3 compatible fashion:

```
from six.moves.urllib.parse import urlencode
```

To have this working on Python 3 as well, please use the `six` library:

```
six.reraise(Exception, "value", tb)
```

- A new check was added to warn about using unicode raw string literals. This is a syntax error in Python 3:

```
a = ur'...'
```

- Added a new *deprecated-sys-function* check, emitted when accessing removed *sys* members.
- Added *xreadlines-attribute* check, emitted when the *xreadlines()* attribute is accessed on a file object.
- Added two new Python 3 porting checks, *exception-escape* and *comprehension-escape*

  These two are emitted whenever pylint detects that a variable defined in the said blocks is used outside of the given block. On Python 3 these values are deleted.

```
try:
  1/0
except ZeroDivisionError as exc:
   ...
print(exc) # This will raise a NameError on Python 3

[i for i in some_iterator if some_condition(i)]
print(i) # This will raise a NameError on Python 3
```

9.1.3 Other Changes

- *defaultdict* and subclasses of *dict* are now handled for *dict-iter-** checks. That means that the following code will now emit warnings for when *iteritems* and friends are accessed:

```
some_dict = defaultdict(list)
...
some_dict.iterkeys()
```

- Enum classes no longer trigger *too-few-methods*
- Special methods now count towards *too-few-methods*, and are considered part of the public API. They are still not counted towards the number of methods for *too-many-methods*.
- docparams allows abstract methods to document returns documentation even if the default implementation does not return something. They also no longer need to document raising a NotImplementedError.

9.2 What's New In Pylint 1.8

Release 1.8

Date 2017-12-15

9.2.1 Summary – Release highlights

- None so far

9.2.2 New checkers

- A new check was added, `shallow-copy-environ`.

  This warning message is emitted when shallow copy of os.environ is created. Shallow copy of os.environ doesn't work as people may expect. os.environ is not a dict object but rather a proxy object, so any changes made on copy may have unexpected effects on os.environ

Instead of copy.copy(os.environ) method os.environ.copy() should be used.

See https://bugs.python.org/issue15373 for details.

```
import copy
import os
wrong_env_copy = copy.copy(os.environ)  # will emit pylint warning
wrong_env_copy['ENV_VAR'] = 'new_value'  # changes os.environ
assert os.environ['ENV_VAR'] == 'new_value'

good_env_copy = os.environ.copy()  # the right way
good_env_copy['ENV_VAR'] = 'different_value'  # doesn't change os.environ
assert os.environ['ENV_VAR'] == 'new_value'
```

- A new check was added, `keyword-arg-before-vararg`.

  This warning message is emitted when a function is defined with a keyword argument appearing before variable-length positional arguments (*args). This may lead to args list getting modified if keyword argument's value is not provided in the function call assuming it will take default value provided in the definition.

```
def foo(a, b=3, *args):
    print(a, b, args)

# Case1: a=0, b=2, args=(4,5)
foo(0,2,4,5) # 0 2 (4,5) ==> Observed values are same as expected values

# Case2: a=0, b=<default_value>, args=(4,5)
foo(0,4,5) # 0 4 (5,) ==> args list got modified as well as the observed value of␣
↪b

# Case3: Syntax Error if tried as follows
foo(0,b=2,4,5) # syntax error
```

- A new check was added, `simplify-boolean-expression`.

  This message is emitted when `consider-using-ternary` check would emit not equivalent code, due to truthy element being falsy in boolean context.

```
value = condition and False or other_value
```

  This flawed construct may be simplified to:

```
value = other_value
```

- A new check was added, `bad-thread-instantiation`.

  This message is emitted when the threading.Thread class does not receive the target argument, but receives just one argument, which is by default the group parameter.

  In the following example, the instantiation will fail, which is definitely not desired:

```
import threading
threading.Thread(lambda: print(1)) # Oups, this is the group parameter
```

- A new Python 3 checker was added to warn about accessing functions that have been removed from the itertools module `izip`, `imap`, `iflter`, `izip_longest`, and `ifilterfalse`.

```
from itertools import izip
print(list(izip([1, 2], [3])))
```

Instead use `six.moves` to import a Python 2 and Python 3 compatible function:

```
from six.moves import zip
print(list(zip([1, 2], [3])))
```

- A new Python 3 checker was added to warn about accessing deprecated fields from the types module like `ListType` or `IntType`

```
from types import ListType
print(isinstance([], ListType))
```

Instead use the declarations in the builtin namespace:

```
print(isinstance([], list))
```

- A new Python 3 checker was added to warn about declaring a `next` method that would have implemented the `Iterator` protocol in Python 2 but is now a normal method in Python 3.

```
class Foo(object):
    def next(self):
        return 42
```

Instead implement a `__next__` method and use `six.Iterator` as a base class or alias `next` to `__next__`:

```
class Foo(object):
    def __next__(self):
        return 42
    next = __next__
```

- Three new Python 3 checkers were added to warn about using dictionary methods in non-iterating contexts.

  For example, the following are returning iterators in Python 3:

```
.. code-block:: python
```

  d = {} d.keys()[0] d.items()[0] d.values() + d.keys()

- A new Python 3 porting check was added, `non-ascii-bytes-literals`

  This message is emitted whenever we detect that a bytes string contain non-ASCII characters, which results in a SyntaxError on Python 3.

- A new warning, `raising-format-tuple`, will catch situations where the intent was likely raising an exception with a formatted message string, but the actual code did omit the formatting and instead passes template string and value parameters as separate arguments to the exception constructor. So it detects things like

```
raise SomeError('message about %s', foo)
raise SomeError('message about {}', foo)
```

  which likely were meant instead as

```
raise SomeError('message about %s' % foo)
raise SomeError('message about {}'.format(foo))
```

  This warning can be ignored on projects which deliberately use lazy formatting of messages in all user-facing exception handlers.

- Following the recommendations of PEP479 ,a new Python 3.0 checker was added to warn about raising a `StopIteration` inside a generator. Raising a `StopIteration` inside a generator may be due a direct call to `raise StopIteration`:

```
def gen_stopiter():
    yield 1
    yield 2
    yield 3
    raise StopIteration
```

Instead use a simple `return` statement

```
def gen_stopiter():
    yield 1
    yield 2
    yield 3
    return
```

Raising a `StopIteration` may also be due to the call to `next` function with a generator as argument:

```
def gen_next_raises_stopiter():
    g = gen_ok()
    while True:
        yield next(g)
```

In this case, surround the call to `next` with a try/except block:

```
def gen_next_raises_stopiter():
    g = gen_ok()
    while True:
        try:
            yield next(g)
        except StopIteration:
            return
```

The check about raising a StopIteration inside a generator is also valid if the exception raised inherit from StopIteration. Close #1385

- A new Python checker was added to warn about using a + operator inside call of logging methods when one of the operands is a literal string:

```
import logging
var = "123"
logging.log(logging.INFO, "Var: " + var)
```

Instead use formatted string and positional arguments :

```
import logging
var = "123"
logging.log(logging.INFO, "Var: %s", var)
```

- A new Python checker was added to warn about `inconsistent-return-statements`. A function or a method has inconsistent return statements if it returns both explicit and implicit values :

```
def mix_implicit_explicit_returns(arg):
    if arg < 10:
        return True
    elif arg < 20:
        return
```

According to PEP8, if any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable). Thus, the previous function should be written:

```
def mix_implicit_explicit_returns(arg):
    if arg < 10:
        return True
    elif arg < 20:
        return None
```

Close #1267

9.2.3 Other Changes

- Fixing u" string in superfluous-parens message.
- Configuration options of invalid name checker are significantly redesigned. Predefined rules for common naming styles were introduced. For typical setups, user friendly options like `--function-naming-style=camelCase` may be used in place of hand-written regular expressions. Default linter config enforce PEP8-compatible naming style. See documentation for details.
- Raise meaningful exception in case of invalid reporter class (output format) being selected.
- The docparams extension now allows a property docstring to document both the property and the setter. Therefore setters can also have no docstring.
- The docparams extension now understands property type syntax.

```
class Foo(object):
    @property
    def foo(self):
        """My Sphinx style docstring description.

        :type: int
        """
        return 10
```

```
class Foo(object):
    @property
    def foo(self):
        """int: My Numpy and Google docstring style description."""
        return 10
```

- In case of `--output-format=json`, the dictionary returned holds a new key-value pair. The key is `message-id` and the value the message id.
- Spelling checker has a new configuration parameter `max-spelling-suggestions`, which affects maximum count of suggestions included in emitted message.
- The **invalid-name** check contains the name of the template that caused the failure.

  For the given code, **pylint** used to emit `invalid-name` in the form `Invalid constant name var`, without offering any context why `var` is not such a good name.

  With this change, it is now more clear what should be improved for a name to be accepted according to its corresponding template.

- New configuration flag, `suggestion-mode` was introduced. When enabled, pylint would attempt to emit user-friendly suggestions instead of spurious errors for some known false-positive scenarios. Flag is enabled by default.
- `superfluous-parens` is no longer wrongly emitted for logical statements involving `in` operator (see example below for what used to be false-postive).

```
foo = None
if 'bar' in (foo or {}):
  pass
```

- Redefinition of dummy function is now possible. `function-redefined` message won't be emitted anymore when dummy functions are redefined.
- `missing-param-doc` and `missing-type-doc` are no longer emitted when `Args` and `Keyword Args` are mixed in Google docstring.
- Fix of false positive `useless-super-delegation` message when parameters default values are different from those used in the base class.
- Fix of false positive `useless-else-on-loop` message when break statements are deeply nested inside loop.
- The Python 3 porting checker no longer emits multiple *no-absolute-import* per file.
- The Python 3 porting checker respects disabled checkers found in the config file.
- Modules, classes, or methods consist of compound statements that exceed the `docstring-min-length` are now correctly emitting *missing-docstring*
- Fix no `wrong-import-order` message emitted on ordering of first and third party libraries. With this fix, pylint distinguishes first and third party modules when checking import order.
- Fix the ignored `pylint disable=fixme` directives for comments following the last statement in a file.
- Fix `line-too-long` message deactivated by wrong disable directive. The directive `disable=fixme` doesn't deactivate anymore the emission of `line-too-long` message for long commented lines.
- If the rcfile specified on the command line doesn't exist, then an IOError exception is raised.
- Fix a false positive inconsistent-return-statements message when if statement is inside try/except. (backport from 2.0)
- Fix a false positive inconsistent-return-statements message when while loop are used. (backport from 2.0)
- Fix unused-argument false positives with overshadowed variable in dictionary comprehension. (backport from 2.0)
- Fixing false positive `inconsistent-return-statements` when never returning functions are used (i.e such as sys.exit). (backport from 2.0)
- Fix false positive `inconsistent-return-statements` message when a function is defined under an if statement. (backport from 2.0)
- Fix false positive `inconsistent-return-statements` message by avoiding useless exception inference if the exception is not handled. (backport from 2.0)
- **Fix false positive `undefined-variable` for lambda arguments in** class definitions
- Fix false-positive `undefined-variable` for generated comprehension variables in function default arguments

9.3 What's New In Pylint 1.7

Release 1.7

Date 2017-04-13

9.3.1 Summary – Release highlights

- None yet.

9.3.2 New checkers

- `single-string-used-for-slots` check was added, which is used whenever a class is using a single string as a slot value. While this is technically not a problem per se, it might trip users when manipulating the slots value as an iterable, which would in turn iterate over characters of the slot value. In order to be more straight-forward, always try to use a container such as a list or a tuple for defining slot values.
- We added a new check, `literal-comparison`, which is used whenever **pylint** can detect a comparison to a literal. This is usually not what we want and, potentially, error prone. For instance, in the given example, the first string comparison returns true, since smaller strings are interned by the interpreter, while for larger ones, it will return False:

```
mystring = "ok"
if mystring is "ok": # Returns true
    # do stuff

mystring = "a" * 1000
if mystring is ("a" * 1000): # This will return False
    # do stuff
```

  Instead of using the `is` operator, you should use the == operator for this use case.

- We added a new refactoring message, `consider-merging-isinstance`, which is emitted whenever we can detect that consecutive *isinstance* calls can be merged together. For instance, in this example, we can merge the first two *isinstance* calls:

```
$ cat a.py
if isinstance(x, int) or isinstance(x, float):
    pass
if isinstance(x, (int, float)) or isinstance(x, str):
    pass
$ pylint a.py
R:  1, 0: Consider merging these isinstance calls to isinstance(x, (float, int))␣
↪(consider-merging-isinstance)
R:  3, 0: Consider merging these isinstance calls to isinstance(x, (int, float,␣
↪str)) (consider-merging-isinstance)
```

- A new error check was added, `invalid-metaclass`, which is used whenever *pylint* can detect that a given class is using a metaclass which is invalid for the purpose of the class. This usually might indicate a problem in the code, rather than something done on purpose.

```
# Needs to inherit from *type* in order to be valid
class SomeClass(object):
    ...
```

(continues on next page)

(continued from previous page)

```
class MyClass(metaclass=SomeClass):
    pass
```

- A new warning was added, `useless-super-delegation`, which is used whenever we can detect that an overridden method is useless, relying on *super()* delegation to do the same thing as another method from the MRO.

  For instance, in this example, the first two methods are useless, since they do the exact same thing as the methods from the base classes, while the next two methods are not, since they do some extra operations with the passed arguments.

```
class Impl(Base):

    def __init__(self, param1, param2):
        super(Impl, self).__init__(param1, param2)

    def useless(self, first, second):
        return super(Impl, self).useless(first, second)

    def not_useless(self, first, **kwargs):
        debug = kwargs.pop('debug', False)
        if debug:
            ...
        return super(Impl, self).not_useless(first, **kwargs)

    def not_useless_1(self, first, *args):
        return super(Impl, self).not_useless_1(first + some_value, *args)
```

- A new warning was added, `len-as-condition`, which is used whenever we detect that a condition uses `len(SEQUENCE)` incorrectly. Instead one could use `if SEQUENCE` or `if not SEQUENCE`.

  For instance, all of the examples below:

```
if len(S):
  pass

if not len(S):
  pass

if len(S) > 0:
  pass

if len(S) != 0:
  pass

if len(S) == 0:
  pass
```

  can be written in a more natural way:

```
if S:
  pass

if not S:
  pass
```

  See https://www.python.org/dev/peps/pep-0008/#programming-recommendations for more information.

- A new extension was added, `emptystring.py` which detects whenever we detect comparisons to empty string constants. This extension is disabled by default. For instance, the examples below:

```
if S != "":
  pass

if S == '':
  pass
```

can be written in a more natural way:

```
if S:
  pass

if not S:
  pass
```

An exception to this is when empty string is an allowed value whose meaning is treated differently than `None`. For example the meaning could be user selected no additional options vs. user has not made their selection yet!

You can activate this checker by adding the line:

```
load-plugins=pylint.extensions.emptystring
```

to the `MASTER` section of your `.pylintrc` or using the command:

```
$ pylint a.py --load-plugins=pylint.extensions.emptystring
```

- A new extension was added, `comparetozero.py` which detects whenever we compare integers to zero. This extension is disabled by default. For instance, the examples below:

```
if X != 0:
  pass

if X == 0:
  pass
```

can be written in a more natural way:

```
if X:
  pass

if not X:
  pass
```

An exception to this is when zero is an allowed value whose meaning is treated differently than `None`. For example the meaning could be `None` means no limit, while `0` means the limit it zero!

You can activate this checker by adding the line:

```
load-plugins=pylint.extensions.comparetozero
```

to the `MASTER` section of your `.pylintrc` or using the command:

```
$ pylint a.py --load-plugins=pylint.extensions.comparetozero
```

- We've added new error conditions for `bad-super-call` which now detect the usage of `super(type(self), self)` and `super(self.__class__, self)` patterns. These can lead

to recursion loop in derived classes. The problem is visible only if you override a class that uses these incorrect invocations of `super()`.

For instance, `Derived.__init__()` will correctly call `Base.__init__`. At this point `type(self)` will be equal to `Derived` and the call again goes to `Base.__init__` and we enter a recursion loop.

```
class Base(object):
    def __init__(self, param1, param2):
        super(type(self), self).__init__(param1, param2)

class Derived(Base):
    def __init__(self, param1, param2):
        super(Derived, self).__init__(param1, param2)
```

- The warnings `missing-returns-doc` and `missing-yields-doc` have each been replaced with two new warnings - `missing-[return|yield]-doc` and `missing-[return|yield]-type-doc`. Having these as separate warnings allows the user to choose whether their documentation style requires text descriptions of function return/yield, specification of return/yield types, or both.

```
# This will raise missing-return-type-doc but not missing-return-doc
def my_sphinx_style_func(self):
    """This is a Sphinx-style docstring.

    :returns: Always False
    """
    return False

# This will raise missing-return-doc but not missing-return-type-doc
def my_google_style_func(self):
    """This is a Google-style docstring.

    Returns:
        bool:
    """
    return False
```

- A new refactoring check was added, `redefined-argument-from-local`, which is emitted when **pylint** can detect than a function argument is redefined locally in some potential error prone cases. For instance, in the following piece of code, we have a bug, since the check will never return `True`, given the fact that we are comparing the same object to its attributes.

```
def test(resource):
    for resource in resources:
        # The ``for`` is reusing ``resource``, which means that the following
        # ``resource`` is not what we wanted to check against.
        if resource.resource_type == resource:
           call_resource(resource)
```

  Other places where this check looks are *with* statement name bindings and except handler's name binding.

- A new refactoring check was added, `no-else-return`, which is emitted when pylint encounters an else following a chain of ifs, all of them containing a return statement.

```
def foo1(x, y, z):
    if x:
        return y
    else:  # This is unnecessary here.
        return z
```

We could fix it deleting the `else` statement.

```
def foo1(x, y, z):
    if x:
        return y
    return z
```

- A new Python 3 check was added, `eq-without-hash`, which enforces classes that implement `__eq__` *also* implement `__hash__`. The behavior around classes which implement `__eq__` but not `__hash__` changed in Python 3; in Python 2 such classes would get `object.__hash__` as their default implementation. In Python 3, aforementioned classes get `None` as their implementation thus making them unhashable.

```
class JustEq(object):
   def __init__(self, x):
     self.x = x

   def __eq__(self, other):
     return self.x == other.x

class Neither(object):
  def __init__(self, x):
    self.x = x

class HashAndEq(object):
   def __init__(self, x):
     self.x = x

   def __eq__(self, other):
     return self.x == other.x

   def __hash__(self):
     return hash(self.x)

{Neither(1), Neither(2)}  # OK in Python 2 and Python 3
{HashAndEq(1), HashAndEq(2)}  # OK in Python 2 and Python 3
{JustEq(1), JustEq(2)}  # Works in Python 2, throws in Python 3
```

In general, this is a poor practice which motivated the behavior change.

```
as_set = {JustEq(1), JustEq(2)}
print(JustEq(1) in as_set)  # prints False
print(JustEq(1) in list(as_set))  # prints True
```

In order to fix this error and avoid behavior differences between Python 2 and Python 3, classes should either explicitly set `__hash__` to `None` or implement a hashing function.

```
class JustEq(object):
   def __init__(self, x):
     self.x = x

   def __eq__(self, other):
     return self.x == other.x

   __hash__ = None

{JustEq(1), JustEq(2)}  # Now throws an exception in both Python 2 and Python 3.
```

- 3 new Python 3 checkers were added, `div-method`, `idiv-method` and `rdiv-method`. The magic meth-

ods __div__ and __idiv__ have been phased out in Python 3 in favor of __truediv__. Classes implementing __div__ that still need to be used from Python 2 code not using `from __future__ import division` should implement __truediv__ and alias __div__ to that implementation.

```
from __future__ import division

class DivisibleThing(object):
   def __init__(self, x):
     self.x = x

   def __truediv__(self, other):
     return DivisibleThing(self.x / other.x)

   __div__ = __truediv__
```

- A new Python 3 checker was added to warn about accessing the `message` attribute on Exceptions. The message attribute was deprecated in Python 2.7 and was removed in Python 3. See https://www.python.org/dev/peps/pep-0352/#retracted-ideas for more information.

```
try:
  raise Exception("Oh No!!")
except Exception as e:
  print(e.message)
```

Instead of relying on the `message` attribute, you should explicitly cast the exception to a string:

```
try:
  raise Exception("Oh No!!")
except Exception as e:
  print(str(e))
```

- A new Python 3 checker was added to warn about using `encode` or `decode` on strings with non-text codecs. This check also checks calls to `open` with the keyword argument `encoding`. See https://docs.python.org/3/whatsnew/3.4.html#improvements-to-codec-handling for more information.

```
'hello world'.encode('hex')
```

Instead of using the `encode` method for non-text codecs use the `codecs` module.

```
import codecs
codecs.encode('hello world', 'hex')
```

- A new warning was added, `overlapping-except`, which is emitted when an except handler treats two exceptions which are *overlapping*. This means that one exception is an ancestor of the other one or it is just an alias.

  For example, in Python 3.3+, IOError is an alias for OSError. In addition, socket.error is an alias for OSError. The intention is to find cases like the following:

```
import socket
try:
    pass
except (ConnectionError, IOError, OSError, socket.error):
    pass
```

- A new Python 3 checker was added to warn about accessing `sys.maxint`. This attribute was removed in Python 3 in favor of `sys.maxsize`.

```
import sys
print(sys.maxint)
```

Instead of using `sys.maxint`, use `sys.maxsize`

```
import sys
print(sys.maxsize)
```

- A new Python 3 checker was added to warn about importing modules that have either moved or been removed from the standard library.

  One of the major undertakings with Python 3 was a reorganization of the standard library to remove old or supplanted modules and reorganize some of the existing modules. As a result, roughly 100 modules that exist in Python 2 no longer exist in Python 3. See https://www.python.org/dev/peps/pep-3108/ and https://www.python.org/dev/peps/pep-0004/ for more information. For suggestions on how to handle this, see https://pythonhosted.org/six/#module-six.moves or http://python3porting.com/stdlib.html.

```
from cStringIO import StringIO
```

Instead of directly importing the deprecated module, either use `six.moves` or a conditional import.

```
from six.moves import cStringIO as StringIO

if sys.version_info[0] >= 3:
    from io import StringIO
else:
    from cStringIO import StringIO
```

This checker will assume any imports that happen within a conditional or a `try/except` block are valid.

- A new Python 3 checker was added to warn about accessing deprecated functions on the string module. Python 3 removed functions that were duplicated from the builtin `str` class. See https://docs.python.org/2/library/string.html#deprecated-string-functions for more information.

```
import string
print(string.upper('hello world!'))
```

Instead of using `string.upper`, call the `upper` method directly on the string object.

```
"hello world!".upper()
```

- A new Python 3 checker was added to warn about calling `str.translate` with the removed `deletechars` parameter. `str.translate` is frequently used as a way to remove characters from a string.

```
'hello world'.translate(None, 'low')
```

Unfortunately, there is not an idiomatic way of writing this call in a 2and3 compatible way. If this code is not in the critical path for your application and the use of `translate` was a premature optimization, consider using `re.sub` instead:

```
import re
chars_to_remove = re.compile('[low]')
chars_to_remove.sub('', 'hello world')
```

If this code is in your critical path and must be as fast as possible, consider declaring a helper method that varies based upon Python version.

```
if six.PY3:
    def _remove_characters(text, deletechars):
        return text.translate({ord(x): None for x in deletechars})
else:
    def _remove_characters(text, deletechars):
        return text.translate(None, deletechars)
```

- A new refactoring check was added, `consider-using-ternary`, which is emitted when pylint encounters constructs which were used to emulate ternary statement before it was introduced in Python 2.5.

```
value = condition and truth_value or false_value
```

  Warning can be fixed by using standard ternary construct:

```
value = truth_value if condition else false_value
```

- A new refactoring check was added, `trailing-comma-tuple`, which is emitted when pylint finds an one-element tuple, created by a stray comma. This can suggest a potential problem in the code and it is recommended to use parantheses in order to emphasise the creation of a tuple, rather than relying on the comma itself.

  The warning is emitted for such a construct:

```
a = 1,
```

  The warning can be fixed by adding parantheses:

```
a = (1, )
```

- Two new check were added for detecting an unsupported operation over an instance, `unsupported-assignment-operation` and `unsupported-delete-operation`. The first one is emitted whenever an object does not support item assignment, while the second is emitted when an object does not support item deletion:

```
class A:
    pass
instance = A()
instance[4] = 4 # unsupported-assignment-operation
del instance[4] # unsupported-delete-operation
```

- A new check was added, `relative-beyond-top-level`, which is emitted when a relative import tries to access too many levels in the current package.
- A new check was added, `trailing-newlines`, which is emitted when a file has trailing new lines.
- `invalid-length-returned` check was added, which is emitted when a `__len__` implementation does not return a non-negative integer.
- There is a new extension, `pylint.extensions.mccabe`, which can be used for computing the McCabe complexity of classes and functions.

  You can enable this extension through `--load-plugins=pylint.extensions.mccabe`
- A new check was added, `used-prior-global-declaration`. This is emitted when a name is used prior a global declaration, resulting in a SyntaxError in Python 3.6.
- A new message was added, `assign-to-new-keyword`. This is emitted when used name is known to become a keyword in future Python release. Assignments to keywords would result in `SyntaxError` after switching to newer interpreter version.

```
# While it's correct in Python 2.x, it raises a SyntaxError in Python 3.x
True = 1
False = 0

# Same as above, but it'll be a SyntaxError starting from Python 3.7
async = "async"
await = "await
```

9.3.3 Other Changes

- We don't emit by default `no-member` if we have opaque inference objects in the inference results

  This is controlled through the new flag `--ignore-on-opaque-inference`, which is by default True. The inference can return multiple potential results while evaluating a Python object, but some branches might not be evaluated, which results in partial inference. In that case, it might be useful to still emit no-member and other checks for the rest of the inferred objects.

- Namespace packages are now supported by pylint. This includes both explicit namespace packages and implicit namespace packages, supported in Python 3 through PEP 420.

- A new option was added, `--analyse-fallback-block`.

  This can be used to support both Python 2 and 3 compatible import block code, which means that the import block might have code that exists only in one or another interpreter, leading to false positives when analysed. By default, this is false, you can enable the analysis for both branches using this flag.

- `ignored-argument-names` option is now used for ignoring arguments for unused-variable check.

  This option was used for ignoring arguments when computing the correct number of arguments a function should have, but for handling the arguments with regard to unused-variable check, dummy-variables-rgx was used instead. Now, ignored-argument-names is used for its original purpose and also for ignoring the matched arguments for the unused-variable check. This offers a better control of what should be ignored and how. Also, the same option was moved from the design checker to the variables checker, which means that the option now appears under the `[VARIABLES]` section inside the configuration file.

- A new option was added, `redefining-builtins-modules`, for controlling the modules which can redefine builtins, such as six.moves and future.builtins.

- A new option was added, `ignore-patterns`, which is used for building a blacklist of directories and files matching the regex patterns, similar to the `ignore` option.

- The reports are now disabled by default, as well as the information category warnings.

- `arguments-differ` check was rewritten to take in consideration keyword only parameters and variadics.

  Now it also complains about losing or adding capabilities to a method, by introducing positional or keyword variadics. For instance, *pylint* now complains about these cases:

```
class Parent(object):

    def foo(self, first, second):
        ...

    def bar(self, **kwargs):
        ...

    def baz(self, *, first):
        ...
```

(continues on next page)

(continued from previous page)

```
class Child(Parent):

    # Why subclassing in the first place?
    def foo(self, *args, **kwargs):
        # mutate args or kwargs.
        super(Child, self).foo(*args, **kwargs)

    def bar(self, first=None, second=None, **kwargs):
        # The overridden method adds two new parameters,
        # which can also be passed as positional arguments,
        # breaking the contract of the parent's method.

    def baz(self, first):
        # Not keyword-only
```

- `redefined-outer-name` is now also emitted when a nested loop's target variable is the same as an outer loop.

```
for i, j in [(1, 2), (3, 4)]:
    for j in range(i):
        print(j)
```

- relax character limit for method and function names that starts with _. This will let people to use longer descriptive names for methods and functions with a shorter scope (considered as private). The same idea applies to variable names, only with an inverse rule: you want long descriptive names for variables with bigger scope, like globals.
- Add `InvalidMessageError` exception class and replace `assert` in pylint.utils with `raise InvalidMessageError`.
- `UnknownMessageError` (formerly `UnknownMessage`) and `EmptyReportError` (formerly `EmptyReport`) are now provided by the new `pylint.exceptions` submodule instead of `pylint.utils` as before.
- We now support inline comments for comma separated values in the configurations

  For instance, you can now use the **#** sign for having comments inside comma separated values, as seen below:

```
disable=no-member, # Don't care about it for now
        bad-indentation, # No need for this
        import-error
```

  Of course, interweaving comments with values is also working:

```
disable=no-member,
        # Don't care about it for now
        bad-indentation # No need for this
```

  This works by setting the inline comment prefixes accordingly.

- Added epytext docstring support to the docparams extension.
- We added support for providing hints when not finding a missing member.

  For example, given the following code, it should be obvious that the programmer intended to use the `mail` attribute, rather than `email`.

```
class Contribution:
    def __init__(self, name, email, date):
        self.name = name
        self.mail = mail
        self.date = date

for c in contributions:
    print(c.email) # Oups
```

pylint will now warn that there is a chance of having a typo, suggesting new names that could be used instead.

```
$ pylint a.py
E: 8,10: Instance of 'Contribution' has no 'email' member; maybe 'mail'?
```

The behaviour is controlled through the `--missing-member-hint` option. Other options that come with this change are `--missing-member-max-choices` for choosing the total number of choices that should be picked in this situation and `--missing-member-hint-distance`, which specifies a metric for computing the distance between the names (this is based on Levenshtein distance, which means the lower the number, the more pickier the algorithm will be).

- `PyLinter.should_analyze_file` has a new parameter, `is_argument`, which specifies if the given path is a **pylint** argument or not.

  `should_analyze_file` is called whenever **pylint** tries to determine if a file should be analyzed, defaulting to files with the `.py` extension, but this function gets called only in the case where the said file is not passed as a command line argument to **pylint**. This usually means that pylint will analyze a file, even if that file has a different extension, as long as the file was explicitly passed at command line. Since `should_analyze_file` cannot be overridden to handle all the cases, the check for the provenience of files was moved into `should_analyze_file`. This means we now can write something similar with this example, for ignoring every file respecting the desired property, disregarding the provenience of the file, being it a file passed as CLI argument or part of a package.

```
from pylint.lint import Run, PyLinter

class CustomPyLinter(PyLinter):

     def should_analyze_file(self, modname, path, is_argument=False):
         if respect_condition(path):
             return False
         return super().should_analyze_file(modname, path, is_argument=is_
↪argument)


class CustomRun(Run):
     LinterClass = CustomPyLinter

CustomRun(sys.argv[1:])
```

- Imports aliased with underscore are skipped when checking for unused imports.
- `bad-builtin` and `redefined-variable-type` are now extensions, being disabled by default. They can be enabled through: `--load-plugins=pylint.extensions.redefined_variable_type, pylint.extensions.bad_builtin`
  - Imports checker supports new switch `allow-wildcard-with-all` which disables warning on wildcard import when imported module defines *__all__* variable.
- `differing-param-doc` is now used for the differing part of the old `missing-param-doc`, and

`differing-type-doc` for the differing part of the old `missing-type-doc`.

9.3.4 Bug fixes

- Fix a false positive of `redundant-returns-doc`, occurred when the documented function was using *yield* instead of *return*.
- Fix a false positive of `missing-param-doc` and `missing-type-doc`, occurred when a class docstring uses the `For the parameters, see` magic string but the class `__init__` docstring does not, or vice versa.
- Added proper exception type inference for `missing-raises-doc`. Now:

```
def my_func():
    """My function."""
    ex = ValueError('foo')
    raise ex
```

  will properly be flagged for missing documentation of `:raises ValueError:` instead of `:raises ex:`, among other scenarios.

- Fix false positives of `missing-[raises|params|type]-doc` due to not recognizing valid keyword synonyms supported by Sphinx.
- More thorough validation in `MessagesStore.register_messages()` to detect conflicts between a new message and any existing message id, symbol, or `old_names`.
- We now support having plugins that shares the same name and with each one providing options.

  A plugin can be logically split into multiple classes, each class providing certain capabilities, all of them being tied under the same name. But when two or more such classes are also adding options, then **pylint** crashed, since it already added the first encountered section. Now, these should work as expected.

```
from pylint.checkers import BaseChecker


class DummyPlugin1(BaseChecker):
    name = 'dummy_plugin'
    msgs = {'I9061': ('Dummy short desc 01', 'dummy-message-01', 'Dummy long desc
↪')}
    options = (
        ('dummy_option_1', {
            'type': 'string',
            'metavar': '<string>',
            'help': 'Dummy option 1',
        }),
    )


class DummyPlugin2(BaseChecker):
    name = 'dummy_plugin'
    msgs = {'I9060': ('Dummy short desc 02', 'dummy-message-02', 'Dummy long desc
↪')}
    options = (
        ('dummy_option_2', {
            'type': 'string',
            'metavar': '<string>',
            'help': 'Dummy option 2',
```

(continues on next page)

(continued from previous page)

```
        }),
    )


def register(linter):
    linter.register_checker(DummyPlugin1(linter))
    linter.register_checker(DummyPlugin2(linter))
```

- We do not yield `unused-argument` for singledispatch implementations and do not warn about `function-redefined` for multiple implementations with same name.

```
from functools import singledispatch

@singledispatch
def f(x):
    return 2*x

@f.register(str)
def _(x):
    return -1

@f.register(int)
@f.register(float)
def _(x):
    return -x
```

- `unused-variable` checker has new functionality of warning about unused variables in global module namespace. Since globals in module namespace may be a part of exposed API, this check is disabled by default. For enabling it, set `allow-global-unused-variables` option to false.
- Fix a false-positive `logging-format-interpolation` message, when format specifications are used in formatted string. In general, these operations are not always convertible to old-style formatting used by logging module.
- Added a new switch `single-line-class-stmt` to allow single-line declaration of empty class bodies (as seen in the example below). Pylint won't emit a `multiple-statements` message when this option is enabled.

```
class MyError(Exception): pass
```

  - *too-many-format-args* and *too-few-format-args* are emitted correctly (or not emitted at all, when exact count of elements in RHS cannot be inferred) when starred expressions are used in RHS tuple. For example, code block as shown below detects correctly that the used tuple has in fact three elements, not two.

```
meat = ['spam', 'ham']
print('%s%s%s' % ('eggs', *meat))
```

- *cyclic-import* checker supports local disable clauses. When one of cycle imports was done in scope where disable clause was active, cycle is not reported as violation.

9.3.5 Removed Changes

- `pylint-gui` was removed, because it was deemed unfit for being included in *pylint*. It had a couple of bugs and misfeatures, its usability was subpar and since its development was neglected, we decided it is best to move on without it.

- The HTML reporter was removed, including the `--output-format=html` option. It was lately a second class citizen in Pylint, being mostly neglected. Since we now have the JSON reporter, it can be used as a basis for building more prettier HTML reports than what Pylint can currently generate. This is part of the effort of removing cruft from Pylint, by removing less used features.
- The `--files-output` option was removed. While the same functionality cannot be easily replicated, the JSON reporter, for instance, can be used as a basis for generating the messages per each file.
- `--required-attributes` option was removed.
- `--ignore-iface-methods` option was removed.
- The `--optimize-ast` flag was removed.

  The option was initially added for handling pathological cases, such as joining too many strings using the addition operator, which was leading pylint to have a recursion error when trying to figure out what the string was. Unfortunately, we decided to ignore the issue, since the pathological case would have happen when the code was parsed by Python as well, without actually reaching the runtime step and as such, we decided to remove the error altogether.
- `epylint.py_run`'s *script* parameter was removed.

  Now `epylint.py_run` is always using the underlying `epylint.lint` method from the current interpreter. This avoids some issues when multiple instances of **pylint** are installed, which means that `epylint.py_run` might have ran a different `epylint` script than what was intended.

9.4 What's New In Pylint 1.6

Release 1.6.0

Date 2016-07-07

9.4.1 Summary – Release highlights

Nothing major.

9.4.2 New checkers

- We added a new recommendation check, `consider-iterating-dictionary`, which is emitted when a dictionary is iterated by using `.keys()`.

  For instance, the following code would trigger this warning, since the dictionary's keys can be iterated without calling the method explicitly.

```
for key in dictionary.keys():
    ...

# Can be refactored to:
for key in dictionary:
    ...
```

- `trailing-newlines` check was added, which is emitted when a file has trailing newlines.
- `invalid-length-returned` check was added, which is emitted when the `__len__` special method returns something else than a non-negative number. For instance, this example is triggering it:

```
class Container(object):
    def __len__(self):
        return self._items # Oups, forgot to call len() over it.
```

- Add a new check to the *check_docs* extension for looking for duplicate constructor parameters in a class constructor docstring or in a class docstring.

  The check `multiple-constructor-doc` is emitted when the parameter is documented in both places.

- We added a new extension plugin, `pylint.extensions.mccabe`, which can be used for warning about the complexity in the code.

  You can enable it as in:

```
$ pylint module_or_project --load-plugins=pylint.extensions.mccabe
```

  See more at *Design checker*

9.4.3 New features

- `generated-members` now supports qualified names through regular expressions.

  For instance, for ignoring all the errors generated by `numpy.core`'s attributes, we can now use:

```
$ pylint a.py --generated-members=numpy.*
```

- Add the ability to ignore files based on regex matching, with the new `--ignore-patterns` option.

  Rather than clobber the existing `ignore` option, we decided to have a separate option for it. For instance, for ignoring all the test files, we can now use:

```
$ pylint myproject --ignore-patterns=test.*?py
```

- We added a new option, `redefining-builtins-modules`, which is used for defining the modules which can redefine builtins. *pylint* will emit an error when a builtin is redefined, such as defining a variable called `next`. But in some cases, the builtins can be redefined in the case they are imported from other places, different than the `builtins` module, such is the case for `six.moves`, which contains more forward-looking functions:

```
$ cat a.py
# Oups, now pylint emits a redefined-builtin message.
from six.moves import open
$ pylint a.py --redefining-builtins-modules=six.moves
```

  Default values: `six.moves,future.builtins`

9.4.4 Bug fixes

- Fixed a bug where the top name of a qualified import was detected as an unused variable.
- We don't warn about `invalid-sequence-index` if the indexed object has unknown base classes, that Pylint cannot deduce.

9.4.5 Other Changes

- The `bad-builtin` check was moved into an extension.

The check was complaining about used builtin functions which were supposed to not be used. For instance, `map` and `filter` were falling into this category, since better alternatives can be used, such as list comprehensions. But the check was annoying, since using `map` or `filter` can have its use cases and as such, we decided to move it to an extension check instead. It can now be enabled through `--load-plugins=pylint.extensions.bad_builtin`.

- We use the `configparser` backport internally, for Python 2.

  This allows having comments inside list values, in the configuration, such as:

```
disable=no-member,
        # Don't like this check
        bad-indentation
```

- We now use the isort package internally.

  This improves the `` `wrong-import-order `` check, so now we should have less false positives regarding the import order.

- We do not emit `import-error` or `no-name-in-module` for fallback import blocks by default.

  A fallback import block can be considered a TryExcept block, which contains imports in both branches, such as:

```
try:
    import urllib.request as request
except ImportError:
    import urllib2 as request
```

  In the case where **pylint** can not find one import from the `except` branch, then it will emit an `import-error`, but this gets cumbersome when trying to write compatible code for both Python versions. As such, we don't check these blocks by default, but the analysis can be enforced by using the new `--analyse-fallback-block` flag.

- `reimported` is emitted when the same name is imported from different module, as in:

```
from collections import deque, OrderedDict, deque
```

9.4.6 Deprecated features

- The HTML support was deprecated and will be eventually removed in Pylint 1.7.0.

  This feature was lately a second class citizen in Pylint, being often neglected and having a couple of bugs. Since we now have the JSON reporter, this can be used as a basis for more prettier HTML outputs than what Pylint can currently offer.

- The `--files-output` option was deprecated and will be eventually removed in Pylint 1.7.0.

- The `--optimize-ast` option was deprecated and will be eventually removed in Pylint 1.7.0.

  The option was initially added for handling pathological cases, such as joining too many strings using the addition operator, which was leading pylint to have a recursion error when trying to figure out what the string was. Unfortunately, we decided to ignore the issue, since the pathological case would have happen when the code was parsed by Python as well, without actually reaching the runtime step and as such, we will remove the option in the future.

- The `check_docs` extension is now deprecated. The extension is still available under the `docparams` name, so this should work:

```
$ pylint module_or_package --load-extensions=pylint.extensions.docparams
```

The old name is still kept for backward compatibility, but it will be eventually removed.

9.4.7 Removed features

- None yet

The "Changelog" contains *all* nontrivial changes to Pylint for the current version.

9.5 Pylint's ChangeLog

9.5.1 What's New in Pylint 1.9.5?

Release date: 2019-07-16

- Fix a false positive *invalid name* message when method or attribute name is longer then 30 characters.

  Close #2047
- Fix compatibility with changes to stdlib tokenizer.

9.5.2 What's New in Pylint 1.9.4?

Release date: 2018-12-27

- `deprecated-method` is correctly emitted for `assertEquals` and friends.

  Close #2226
- Fixed false positive when a numpy Attributes section follows a Parameters section

  Close #1867
- Add support for nupmydoc optional return value names.

  Close #2030
- Can specify a default docstring type for when the check cannot guess the type

  Close #1169
- Fix false-postive undefined-variable in nested lambda

  Close #760

9.5.3 What's New in Pylint 1.9.3?

Release date: 2018-07-24

- *chain.from_iterable* no longer emits *dict-{}-not-iterating* when dealing with dict values and keys
- Fix incorrect file path when file absolute path contains multiple `path_strip_prefix` strings.

  Close #1120 and #2280

- *in* is considered iterating context for some of the Python 3 porting checkers

  Close #2186

- Add *–exit-zero* option for continuous integration scripts to more easily call Pylint in environments that abort when a program returns a non-zero (error) status code.

  Close #2042

- [].extend and similar builtin operations don't emit *dict-*-not-iterating* with the Python 3 porting checker

  Close #2187

9.5.4 What's New in Pylint 1.9.2?

Release date: 2018-06-06

- Fix *KeyError* raised when using docparams and NotImplementedError is documented.
- Don't include excepthandlers that don't have a name when looking for exception-escape
- Rewrite the comprehension-escape and exception-escape to work only on the corresponding nodes

These two checks were implemented in terms of visit_namne, that is, for every name in the tree, we looked to see if that name leaked. This was resulting in a couple of false positives and also in a performance issue, since we were calling nodes_of_class() and scope() for each name node. Instead, this approach uses the visit methods for exception and comprehension nodes and looks to see from there if the current scope uses leaked names. This is not the ideal situation as well, ideal would be to have access to the definition frame of each name, but that requires some extra engineering effort in astroid to get it right. Close #2106

- builtins is allowed to redefine builtins. Close #1966

9.5.5 What's New in Pylint 1.9.1?

Release date: 2018-05-14

- Fixed old-raise-syntax always erroring in Python 3.

9.5.6 What's New in Pylint 1.9.0?

Release date: 2018-05-15

- Added two new Python 3 porting checks, *exception-escape* and *comprehension-escape*

  These two are emitted whenever pylint detects that a variable defined in the said blocks is used outside of the given block. On Python 3 these values are deleted.

- Added a new *deprecated-sys-function*, emitted when accessing removed sys members.
- Added *xreadlines-attribute*, emitted when the *xreadlines()* attribute is accessed.
- The Python 3 porting mode can now run with Python 3 as well.
- docparams extension allows abstract methods to document what overriding implementations should return, and to raise NotImplementedError without documenting it.

  Closes #2044

- Special methods do not count towards *too-few-methods*, and are considered part of the public API.

- Enum classes do not trigger *too-few-methods*

  Close #605

- defaultdict and subclasses of dict are now handled for dict-iter-* checks

  Close #2005

- Added a new Python 2/3 check for accessing *operator.div*, which is removed in Python 3

  Close #1936

- Added a new Python 2/3 check for accessing removed urllib functions

  Close #1997

- Added a new check for catching unicode raw string literals

  Close #1938

9.5.7 What's New in Pylint 1.8.4?

Release date: 2018-04-05

- Stop early when we already detected a disable for a line too long

Close #1974

- Add missing string functions to the Python 3 porting checker
- Require no-context-manager to infer paths before walking the tree

Fixes #1874

9.5.8 What's New in Pylint 1.8.3?

Release date: 2018-03-16

- Fix false positive `inconsistent-return-statements` message when a function is defined under an if statement.

  Close #1794

  - Exempt *__doc__* from triggering a *redefined-builtin*

    __doc__ can be used to specify a docstring for a module without passing it as a first-statement string.

- Don't crash on invalid strings when checking for *logging-format-interpolation*

  Close #1944

- Fix false positive `inconsistent-return-statements` message by avoiding useless exception inference if the exception is not handled.

  Close #1794 (second part)

- **Fix false positive undefined-variable for lambda argument in** class definitions

  Close #1824

- Fix false-positive `undefined-variable` for generated comprehension variables in function default arguments

  Close #1897

9.5.9 What's New in Pylint 1.8.2?

Release date: 2018-01-23

- Fixed a crash which occurred when *Uninferable* wasn't properly handled in *stop-iteration-return*

  Close #1779

- Use the proper node to get the name for redefined functions (#1792)

  Close #1774

- Don't crash when encountering bare raises while checking inconsistent returns

  Close #1773

- Fix a false positive `inconsistent-return-statements` message when if statement is inside try/except.

  Close #1770

- Fix a false positive `inconsistent-return-statements` message when while loop are used.

  Close #1772

- Fix `unused-argument` false positives with overshadowed variable in dictionary comprehension.

  Close #1731

- Fix false positive `inconsistent-return-statements` message when never returning functions are used (i.e sys.exit for example).

  Close #1771

- Fix error when checking if function is exception, as in `bad-exception-context`.

9.5.10 What's New in Pylint 1.8.1?

Release date: 2017-12-15

- Wrong version number in __pkginfo__.

9.5.11 What's New in Pylint 1.8?

Release date: 2017-12-15

- Respect disable=... in config file when running with –py3k.
- New warning *shallow-copy-environ* added

  Shallow copy of os.environ doesn't work as people may expect. os.environ is not a dict object but rather a proxy object, so any changes made on copy may have unexpected effects on os.environ

  Instead of copy.copy(os.environ) method os.environ.copy() should be used.

  See https://bugs.python.org/issue15373 for details.

  Close #1301

- Do not display no-absolute-import warning multiple times per file.
- ***trailing-comma-tuple* refactor check now extends to assignment with** more than one element (such as lists)

  Close #1713

- Fixing u" string in superfluous-parens message

  Close #1420

- *abstract-class-instantiated* is now emitted for all inference paths.

  Close #1673

- Add set of predefined naming style to ease configuration of checking naming conventions.

  Closes #1013

- Added a new check, `keyword-arg-before-vararg`

  This is emitted for function definitions in which keyword arguments are placed before variable positional arguments (*args).

  This may lead to args list getting modified if keyword argument's value is not provided in the function call assuming it will take default value provided in the definition.

- The *invalid-name* check contains the name of the template that caused the failure

  Close #1176

- Using the -j flag won't start more child linters than needed.

  Contributed by Roman Ivanov in #1614

- Fix a false positive with bad-python3-import on relative imports

  Close #1608

- Added a new Python 3 check, `non-ascii-bytes-literals`

  Close #1545

- Added a couple of new Python 3 checks for accessing dict methods in non-iterable context

- Protocol checks (not-a-mapping, not-an-iterable and co.) aren't emitted on classes with dynamic getattr

- Added a new warning, 'bad-thread-instantiation'

  This message is emitted when the threading.Thread class does not receive the target argument, but receives just one argument, which is by default the group parameter.

  Close #1327

- In non-quiet mode, absolute path of used config file is logged to standard error. Close #1519

- Raise meaningful exception for invalid reporter class being selected

  When unknown reporter class will be selected as Pylint reporter, meaningful error message would be raised instead of bare `ImportError` or `AttribueError` related to module or reporter class being not found. Close #1388

- Added a new Python 3 check for accessing removed functions from itertools like `izip` or `ifilterfalse`

- Added a new Python 3 check for accessing removed fields from the types module like `UnicodeType` or `XRangeType`

- Added a new Python 3 check for declaring a method `next` that would have been treated as an iterator in Python 2 but a normal function in Python 3.

- Added a new key-value pair in json output. The key is `message-id` and the value is the message id. Close #1512

- Added a new Python 3.0 check for raising a StopIteration inside a generator. The check about raising a StopIteration inside a generator is also valid if the exception raised inherit from StopIteration. Close #1385
- Added a new warning, `raising-format-tuple`, to detect multi-argument exception construction instead of message string formatting.
- Added a new check for method of logging module that concatenate string via + operator Close #1479
- Added parameter for limiting number of suggestions in spellchecking checkers
- Fix a corner-case in `consider-using-ternary` checker.

  When object `A` used in `X and A or B` was falsy in boolean context, Pylint incorrectly emitted non-equivalent ternary-based suggestion. After a change message is correctly not emitted for this case. Close #1559
- Added `suggestion-mode` configuration flag. When flag is enabled, informational message is emitted instead of cryptic error message for attributes accessed on c-extensions. Close #1466
- Fix a false positive `useless-super-delegation` message when parameters default values are different from those used in the base class. Close #1085
- Disabling 'wrong-import-order', 'wrong-import-position', or 'ungrouped-imports' for a single line now prevents that line from triggering violations on subsequent lines.

  Close #1336
- Added a new Python check for inconsistent return statements inside method or function. Close #1267
- Fix `superfluous-parens` false positive related to handling logical statements involving `in` operator.

  Close #574
- `function-redefined` message is no longer emitted for functions and methods which names matches dummy variable name regular expression. Close #1369
- Fix `missing-param-doc` and `missing-type-doc` false positives when mixing `Args` and `Keyword Args` in Google docstring. Close #1409

  - Fix `missing-docstring` false negatives when modules, classes, or methods

  consist of compound statements that exceed the `docstring-min-length`
- Fix `useless-else-on-loop` false positives when break statements are deeply nested inside loop. Close #1661
- Fix no `wrong-import-order` message emitted on ordering of first and third party libraries. With this fix, pylint distinguishes third and first party modules when checking import order. Close #1702
- Fix `pylint disable=fixme` directives ignored for comments following the last statement in a file. Close #1681
- Fix `line-too-long` message deactivated by wrong disable directive. The directive `disable=fixme` doesn't deactivate anymore the emission of `line-too-long` message for long commented lines. Close #1741
- If the rcfile specified on the command line doesn't exist, then an IOError exception is raised. Close #1747

9.5.12 What's New in Pylint 1.7.1?

Release date: 2017-04-17

- Fix a false positive which occurred when an exception was reraised

  Close #1419

- Fix a false positive of `disallow-trailing-tuple`

  The check was improved by verifying for non-terminating newlines, which should exempt function calls and function definitions from the check Close #1424

9.5.13 What's New in Pylint 1.7?

Release date: 2017-04-13

- Don't emit missing-final-newline or trailing-whitespace for formfeeds (page breaks).

  Close #1218 and #1219

- Don't emit by default no-member if we have opaque inference objects in the inference results

  This is controlled through the new flag ignore-on-opaque-inference, which is by default True. The inference can return multiple potential results while evaluating a Python object, but some branches might not be evaluated, which results in partial inference. In that case, it might be useful to still emit no-member and other checks for the rest of the inferred objects.

- Added new message *assign-to-new-keyword* to warn about assigning to names which will become a keyword in future Python releases.

  Close #1351

- Split the 'missing or differing' in parameter documentation in different error. 'differing-param-doc' covers the differing part of the old 'missing-param-doc', and 'differing-type-doc' covers the differing part of the old 'missing-type-doc'

  Close #1342

- Added a new error, 'used-prior-global-declaration', which is emitted when a name is used prior a global declaration in a function. This causes a SyntaxError in Python 3.6

  Close #1257

- The protocol checks are emitting their messages when a special method is set to None.

  Close #1263

- Properly detect if imported name is assigned to same name in different scope.

  Close #636, #848, #851, and #900

- Require one space for annotations with type hints, as per PEP 8.

- 'trailing-comma-tuple' check was added

  This message is emitted when pylint finds an one-element tuple, created by a stray comma. This can suggest a potential problem in the code and it is recommended to use parantheses in order to emphasise the creation of a tuple, rather than relying on the comma itself.

- Don't emit not-callable for instances with unknown bases.

  Close #1213

- Treat keyword only arguments the same as positional arguments with regard to unused-argument check

- Don't try to access variables defined in a separate scope when checking for `protected-access`
- Added new check to detect incorrect usage of len(SEQUENCE) inside test conditions.
- Added new extension to detect comparisons against empty string constants
- Added new extension to detect comparisons of integers against zero
- Added new error conditions for 'bad-super-call'

  Now detects `super(type(self), self)` and `super(self.__class__, self)` which can lead to recursion loop in derived classes.
- PyLinter.should_analyze_file has a new optional parameter, called *is_argument*

  Close #1079
- Add attribute hints for missing members

  Closes #1035
- Add a new warning, 'redefined-argument-from-local'

  Closes #649
- Support inline comments for comma separated values in the config file

  Closes #1024
- epylint.py_run's *script* parameter was removed.
- epylint.py_run now uses `shell=False` for running the underlying process.

  Closes #441
- Added a new warning, 'useless-super-delegation'

  Close 839.
- Added a new error, 'invalid-metaclass', raised when we can detect that a class is using an improper metaclass.

  Close #579
- Added a new refactoring message, 'literal-comparison'.

  Close #786
- arguments-differ takes in consideration kwonlyargs and variadics

  Close #983
- Removed –optimized-ast. Part of #975.
- Removed –files-output option. Part of #975.
- Removed pylint-gui from the package.
- Removed the HTML reporter. Part of #975.
- ignored-argument-names is now used for ignoring arguments for unused-variable check.

  This option was used for ignoring arguments when computing the correct number of arguments a function should have, but for handling the arguments with regard to unused-variable check, dummy-variables-rgx was used instead. Now, ignored-argument-names is used for its original purpose and also for ignoring the matched arguments for the unused-variable check. This offers a better control of what should be ignored and how. Also, the same option was moved from the design checker to the variables checker, which means that the option now appears under the `[VARIABLES]` section inside the configuration file. Close #862.

- Fix a false positive for keyword variadics with regard to keyword only arguments.

  If a keyword only argument was necessary for a function, but that function was called with keyword variadics (**kwargs), then we were emitting a missing-kwoa false positive, which is now fixed.

  Close #934.

- Fix some false positives with unknown sized variadics.

  Close #878

- Added a new extension, check_docstring, for checking PEP 257 conventions.

  Closes #868.

- config files with BOM markers can now be read.

  Close #864.

- epylint.py_run does not crash on big files, using .communicate() instead of .wait()

  Close #599

- Disable reports by default and show the evaluation score by default

  As per discussion from issue #746, the reports were disabled by default in order to simplify the interaction between the tool and the users. The score is still shown by default, as a way of closely measuring when it increases or decreases due to changes brought to the code.

- Disable the information category messages by default.

  This is a step towards making pylint more sane, as per the discussion from issue #746.

- Catch more cases as not proper iterables for __slots__ with regard to invalid-slots pattern. Closes issue #775.

- empty indent strings are rejected.

- Added a new error, 'relative-beyond-top-level', which is emitted when a relative import was attempted beyond the top level package.

  Closes issue #588.

- Added a new warning, 'unsupported-assignment-operation', which is emitted when item assignment is tried on an object which doesn't have this ability. Closes issue #591.

- Added a new warning, 'unsupported-delete-operation', which is emitted when item deletion is tried on an object which doesn't have this ability. Closes issue #592.

- Fix a false positive of 'redundant-returns-doc', occurred when the documented function was using *yield* instead of *return*.

  Closes issue #984.

- Fix false positives of 'missing-[raises|params|type]-doc' due to not recognizing keyword synonyms supported by Sphinx.

- Added a new refactoring message, 'consider-merging-isinstance', which is emitted whenever we can detect that consecutive isinstance calls can be merged together.

  Closes issue #968

- Fix a false positive of 'missing-param-doc' and 'missing-type-doc', occurred when a class docstring uses the 'For the parameters, see' magic string but the class __init__ docstring does not, or vice versa.

- *redefined-outer-name* is now also emitted when a nested loop's target variable is the same as a target variable in an outer loop.

  Closes issue #911.

- Added proper exception type inference for 'missing-raises-doc'.
- Added InvalidMessageError exception class to replace asserts in pylint.utils.
- More thorough validation in MessagesStore.register_messages() to avoid one message accidentally overwriting another.
- InvalidMessageError, UnknownMessage, and EmptyReport exceptions are moved to the new pylint.exceptions submodule.
- UnknownMessage and EmptyReport are renamed to UnknownMessageError and EmptyReportError.
- Warnings 'missing-returns-type-doc' and 'missing-yields-type-doc' have each been split into two warnings - 'missing-[return|yield]-doc' and 'missing-[return|yield]-type-doc'.
- Added epytext support to docparams extension.

  Closes #1029.
- Support having plugins with the same name and with options defined

  Closes #1018
- Sort configuration options in a section

  Closes #1087
- Added a new Python 3 warning around implementing '__div__', '__idiv__', or '__rdiv__' as those methods are phased out in Python 3.
- Added a new warning, 'overlapping-except', which is emitted when two exceptions in the same except-clause are aliases for each other or one exceptions is an ancestor of another.
- Avoid crashing on ill-formatted strings when checking for string formatting errors.
- Added a new Python 3 warning for calling 'str.encode' or 'str.decode' with a non-text encoding.
- Added new coding convention message, 'single-string-used-for-slots'.

  Closes #1166
- Added a new Python 3 check for accessing 'sys.maxint' which was removed in Python 3 in favor of 'sys.maxsize'
- Added a new Python 3 check for bad imports.
- Added a new Python 3 check for accessing deprecated string functions.
- Do not warn about unused arguments or function being redefined in singledispatch registered implementations.

  Closes #1032 and #1034
- Added refactoring message 'no-else-return'.
- Improve unused-variable checker to warn about unused variables in module scope.

  Closes #919
- Ignore modules import as _ when checking for unused imports.

  Closes #1190
- Improve handing of Python 3 classes with metaclasses declared in nested scopes.

  Closes #1177
- Added refactoring message 'consider-using-ternary'.

  Closes #1204

- Bug-fix for false-positive logging-format-interpolation‘ when format specifications are used in formatted string.

  Fixes #572

- Added a new switch `single-line-class-stmt` to allow single-line declaration of empty class bodies.

  Closes #738

- Protected access in form *type(self)._attribute* are now allowed.

  Fixes #1031

- Let the user modify msg-template when Pylint is called from a Python script

  Fixes #1269

- Imports checker supports new switch `allow-wildcard-with-all` which disables warning on wildcard import when imported module defines *__all__* variable.

  Fixes #831

- *too-many-format-args* and *too-few-format-args* are emitted correctly when starred expression are used in RHS tuple.

  Fixes #957

- *cyclic-import* checker supports local disable clauses. When one of cycle imports was done in scope where disable clause was active, cycle is not reported as violation.

  Fixes #59

9.5.14 What's new in Pylint 1.6.3?

Release date: 2016-07-18

- Do not crash when inferring uninferable exception types for docparams extension

  Close #998

9.5.15 What's new in Pylint 1.6.2?

Release date: TBA

- Do not crash when printing the help of options with default regular expressions

  Close #990

- More granular versions for deprecated modules.

  Close #991

9.5.16 What's new in Pylint 1.6.1?

Release date: 2016-07-07

- Use environment markers for supporting conditional dependencies.

9.5.17 What's New in Pylint 1.6.0?

Release date: 2016-07-03

- Added a new extension, *pylint.extensions.mccabe*, for warning about complexity in code.
- Deprecate support for –optimize-ast. Part of #975.
- Deprecate support for the HTML output. Part of #975.
- Deprecate support for –output-files. Part of #975.
- Fixed a documentation error for the check_docs extension. Fixes #735.
- Made the list of property-defining decorators configurable.
- Fix a bug where the top name of a qualified import was detected as unused variable.

  Close #923.
- bad-builtin is now an extension check.
- generated-members support qualified name through regular expressions.

  For instance, one can specify a regular expression as –generated-members=astroid.node_classes.* for ignoring every no-member error that is accessed as in *astroid.node_classes.missing.object*.
- Add the ability to ignore files based on regex matching, with the new `--ignore-patterns` option.

  This addresses issue #156 by allowing for multiple ignore patterns to be specified. Rather than clobber the existing ignore option, we introduced a new one called ignore-patterns.
- Added a new error, 'trailing-newlines', which is emitted when a file has trailing new lines.

  Closes issue #682.
- Add a new option, 'redefining-builtins-modules', for controlling the modules which can redefine builtins, such as six.moves and future.builtins.

  Close #464.
- 'reimported' is emitted when the same name is imported from different module.

  Close #162.
- Add a new recommendation checker, 'consider-iterating-dictionary', which is emitted which is emitted when a dictionary is iterated through .keys().

  Close #699
- Use the configparser backport for Python 2

  This fixes a problem we were having with comments inside values, which is fixed in Python 3's configparser. Close #828
- A new error was added, 'invalid-length-returned', when the *__len__* special method returned something else than a non-negative number.

  Close issue #557
- Switch to using isort internally for wrong-import-order.

  Closes #879.
- check_docs extension can find constructor parameters in __init__.

  Closes #887.

- Don't warn about invalid-sequence-index if the indexed object has unknown base classes.

  Closes #867

- Don't crash when checking, for super-init-not-called, a method defined in an if block.
- Do not emit import-error or no-name-in-module for fallback import blocks by default.

  Until now, we warned with these errors when a fallback import block (a TryExcept block that contained imports for Python 2 and 3) was found, but this gets cumbersome when trying to write compatible code. As such, we don't check these blocks by default, but the analysis can be enforced by using the new `--analyse-fallback-block` flag.

  Close #769.

9.5.18 What's New in Pylint 1.5.5?

Release date: 2016-03-21

- Let visit_importfrom from Python 3 porting checker be called when everything is disabled

  Because the visit method was filtering the patterns it was expecting to be activated, it didn't run when everything but one pattern was disabled, leading to spurious false positives

  Close #852

- Don't emit unsubscriptable-value for classes with unknown base classes.

  Close #776.

- Use an OrderedDict for storing the configuration elements

  This fixes an issue related to impredictible order of the disable / enable elements from a config file. In certain cases, the disable was coming before the enable which resulted in classes of errors to be enabled, even though the intention was to disable them. The best example for this was in the context of running multiple processes, each one of it having different enables / disables that affected the output.

  Close #815

- Don't consider bare and broad except handlers as ignoring NameError, AttributeError and similar exceptions, in the context of checkers for these issues.

  Closes issue #826

9.5.19 What's New in Pylint 1.5.4?

Release date: 2016-01-15

- Merge StringMethodChecker with StringFormatChecker. This fixes a bug where disabling all the messages and enabling only a handful of messages from the StringFormatChecker would have resulted in no messages at all.
- Don't apply unneeded-not over sets.

9.5.20 What's New in Pylint 1.5.3?

Release date: 2016-01-11

- Handle the import fallback idiom with regard to wrong-import-order.

  Closes issue #750.

- Decouple the displaying of reports from the displaying of messages

  Some reporters are aggregating the messages instead of displaying them when they are available. The actual displaying was conflatted in the generate_reports. Unfortunately this behaviour was flaky and in the case of the JSON reporter, the messages weren't shown at all if a file had syntax errors or if it was missing. In order to fix this, the aggregated messages can now be displayed with Reporter.display_message, while the reports are displayed with display_reports.

  Closes issues #766 and #765.

- Ignore function calls with variadic arguments without a context.

  Inferring variadic positional arguments and keyword arguments will result into empty Tuples and Dicts, which can lead in some cases to false positives with regard to no-value-for-parameter. In order to avoid this, until we'll have support for call context propagation, we're ignoring such cases if detected. Closes issue #722.

- Treat AsyncFunctionDef just like FunctionDef nodes, by implementing visit_asyncfunctiondef in terms of visit_functiondef.

  Closes issue #767.

- Take in account kwonlyargs when verifying that arguments are defined with the check_docs extension.

  Closes issue #745.

- Suppress reporting 'unneeded-not' inside *__ne__* methods

  Closes issue #749.

9.5.21 What's New in Pylint 1.5.2?

Release date: 2015-12-21

- Don't crash if graphviz is not installed, instead emit a warning letting the user to know.

  Closes issue #168.

- Accept only functions and methods for the deprecated-method checker.

  This prevents a crash which can occur when an object doesn't have .qname() method after the inference.

- Don't emit super-on-old-class on classes with unknown bases. Closes issue #721.

- Allow statements in *if* or *try* blocks containing imports.

  Closes issue #714.

9.5.22 What's New in Pylint 1.5.1?

Release date: 2015-12-02

- Fix a crash which occurred when old visit methods are encountered in plugin modules. Closes issue #711.

- Add wrong-import-position to check_messages's decorator arguments for ImportChecker.leave_module This fixes an esoteric bug which occurs when ungrouped-imports and wrong-import-order are disabled and pylint is executed on multiple files. What happens is that without wrong-import-position in check_messages, leave_module will never be called, which means that the first non-import node from other files might leak into the current file, leading to wrong-import-position being emitted by pylint.

- Fix a crash which occurred when old visit methods are encountered in plugin modules. Closes issue #711.

- Don't emit import-self and cyclic-import for relative imports of modules with the same name as the package itself. Closes issues #708 and #706.

9.5.23 What's New in Pylint 1.5.0?

Release date: 2015-11-29

- Added multiple warnings related to imports. 'wrong-import-order' is emitted when PEP 8 recommendations regarding imports are not respected (that is, standard imports should be followed by third-party imports and then by local imports). 'ungrouped-imports' is emitted when imports from the same package or module are not placed together, but scattered around in the code. 'wrong-import-position' is emitted when code is mixed with imports, being recommended for the latter to be at the top of the file, in order to figure out easier by a human reader what dependencies a module has. Closes issue #692.
- Added a new refactoring warning, 'unneeded-not', emitted when an expression with the not operator could be simplified. Closes issue #670.
- Added a new refactoring warning, 'simplifiable-if-statement', used when an if statement could be reduced to a boolean evaluation of its test. Closes issue #698.
- Added a new refactoring warning, 'too-many-boolean-expressions', used when an if statement contains too many boolean expressions, which makes the code less maintainable and harder to understand. Closes issue #677.
- Property methods are shown as attributes instead of functions in pyreverse class diagrams. Closes Issue #284
- Add a new refactoring error, 'too-many-nested-blocks', which is emitted when a function or a method has too many nested blocks, which makes the code less readable and harder to understand. Closes issue #668.
- Add a new error, 'unsubscriptable-object', that is emitted when value used in subscription expression doesn't support subscription (i.e. doesn't define __getitem__ method).
- Don't warn about abstract classes instantiated in their own body. Closes issue #627.
- Obsolete options are not present by default in the generated configuration file. Closes issue #632.
- non-iterator-returned can detect classes with iterator-metaclasses. Closes issue #679.
- Add a new error, 'unsupported-membership-test', emitted when value to the right of the 'in' operator doesn't support membership test protocol (i.e. doesn't define __contains__/__iter__/__getitem__)
- Add new errors, 'not-an-iterable', emitted when non-iterable value is used in an iterating context (starargs, for-statement, comprehensions, etc), and 'not-a-mapping', emitted when non-mapping value is used in a mapping context. Closes issue #563.
- Make 'no-self-use' checker not emit a warning if there is a 'super()' call inside the method. Closes issue #667.
- Add checker to identify multiple imports on one line. Closes issue #598.
- Fix unused-argument false positive when the "+=" operator is used. Closes issue #518.
- Don't emit import-error for ignored modules. PyLint will not emit import errors for any import which is, or is a subpackage of, a module in the ignored-modules list. Closes issue #223.
- Fix unused-import false positive when the import is used in a class assignment. Closes issue #475
- Add a new error, 'not-context-manager', emitted when something that doesn't implement __enter__ and __exit__ is used in a with statement.
- Add a new warning, 'confusing-with-statement', emitted by the base checker, when an ambiguous looking with statement is used. For example *with open() as first, second* which looks like a tuple assignment but is actually 2 context managers.
- Add a new warning, 'duplicate-except', emitted when there is an exception handler which handles an exception type that was handled before. Closes issue #485.

- A couple of warnings got promoted to errors, since they could uncover potential bugs in the code. These warnings are: assignment-from-none, unbalanced-tuple-unpacking, unpacking-non-sequence, non-iterator-returned. Closes issue #388.
- Allow ending a pragma control with a semicolon. In this way, users can continue a pragma control with a reason for why it is used, as in *# pylint: disable=old-style-class;reason=...*. Closes issue #449.
- –jobs can be used with –load-plugins now. Closes issue #456.
- Improve the performance of –jobs when dealing only with a package name. Closes issue #479.
- Don't emit an unused-wildcard-import when the imported name comes from another module and it is in fact a __future__ name.
- The colorized reporter now works on Windows. Closes issue #96.
- Remove pointless-except warning. It was previously disabled by default and it wasn't very useful. Closes issue #506.
- Fix a crash on Python 3 related to the string checker, which crashed when it encountered a bytes string with a .format method called.
- Don't warn about no-self-use for builtin properties.
- Fix a false positive for bad-reversed-sequence, when a subclass of a *dict* provides a __reversed__ method.
- Change the default no-docstring-rgx so missing-docstring isn't emitted for private functions.
- Don't emit redefined-outer-name for __future__ directives. Closes issue #520.
- Provide some hints for the bad-builtin message. Closes issue #522.
- When checking for invalid arguments to a callable, in typecheck.py, look up for the __init__ in case the found __new__ comes from builtins.

  Since the __new__ comes from builtins, it will not have attached any information regarding what parameters it expects, so the check will be useless. Retrieving __init__ in that case will at least detect a couple of false negatives. Closes issue #429.
- Don't emit no-member for classes with unknown bases.

  Since we don't know what those bases might add, we simply ignore the error in this case.
- Lookup in the implicit metaclass when checking for no-member, if the class in question has an implicit metaclass, which is True for new style classes. Closes issue #438.
- Add two new warnings, duplicate-bases and inconsistent-mro.

  duplicate-bases is emitted when a class has the same bases listed more than once in its bases definition, while inconsistent-mro is emitted when no sane mro hierarchy can be determined. Closes issue #526.
- Remove interface-not-implemented warning. Closes issue #532.
- Remove the rest of interface checks: interface-is-not-class, missing-interface-method, unresolved-interface. The reason is that its better to start recommending ABCs instead of the old Zope era of interfaces. One side effect of this change is that ignore-iface-methods becomes a noop, it's deprecated and it will be removed at some time.
- Emit a proper deprecation warning for reporters.BaseReporter.add_message.

  The alternative way is to use handle_message. add_message will be removed in Pylint 1.6.
- Added new module 'extensions' for optional checkers with the test directory 'test/extensions' and documentation file 'doc/extensions.rst'.
- Added new checker 'extensions.check_docs' that verifies parameter documention in Sphinx, Google, and Numpy style.

- Detect undefined variable cases, where the "definition" of an undefined variable was in del statement. Instead of emitting used-before-assignment, which is totally misleading, it now emits undefined-variable. Closes issue #528.
- Don't emit attribute-defined-outside-init and access-member-before-definition for mixin classes. Actual errors can occur in mixin classes, but this is controlled by the ignore-mixin-members option. Closes issue #412.
- Improve the detection of undefined variables and variables used before assignment for variables used as default arguments to function, where the variable was first defined in the class scope. Closes issue #342 and issue #404.
- Add a new warning, 'unexpected-special-method-signature', which is emitted when a special method (dunder method) doesn't have the expected signature, which can lead to actual errors in the application code. Closes issue #253.
- Remove 'bad-context-manager' due to the inclusion of 'unexpected-special-method-signature'.
- Don't emit no-name-in-module if the import is guarded by an ImportError, Exception or a bare except clause.
- Don't emit no-member if the attribute access node is protected by an except handler, which handles AttributeError, Exception or it is a bare except.
- Don't emit import-error if the import is guarded by an ImportError, Exception or a bare except clause.
- Don't emit undefined-variable if the node is guarded by a NameError, Exception or bare except clause.
- Add a new warning, 'using-constant-test', which is emitted when a conditional statement (If, IfExp) uses a test which is always constant, such as numbers, classes, functions etc. This is most likely an error from the user's part. Closes issue #524.
- Don't emit 'raising-non-exception' when the exception has unknown bases. We don't know what those bases actually are and it's better to assume that the user knows what he is doing rather than emitting a message which can be considered a false positive.
- Look for a .pylintrc configuration file in the current folder, if pylintrc is not found. Dotted pylintrc files will not be searched in the parents of the current folder, as it is done for pylintrc.
- Add a new error, 'invalid-unary-type-operand', emitted when an unary operand is used on something which doesn't support that operation (for instance, using the unary bitwise inversion operator on an instance which doesn't implement __invert__).
- Take in consideration differences between arguments of various type of functions (classmethods, staticmethods, properties) when checking for *arguments-differ*. Closes issue #548.
- astroid.inspector was moved to pylint.pyreverse, since it belongs there and it doesn't need to be in astroid.
- astroid.utils.LocalsVisitor was moved to pylint.pyreverse.LocalsVisitor.
- pylint.checkers.utils.excepts_import_error was removed. Use pylint.chekcers.utils.error_of_type instead.
- Don't emit undefined-all-variables for nodes which can't be inferred (YES nodes).
- yield-outside-func is also emitted for *yield from*.
- Add a new error, 'too-many-star-expressions', emitted when there are more than one starred expression (**x*) in an assignment. The warning is emitted only on Python 3.
- Add a new error, 'invalid-star-assignment-target', emitted when a starred expression (**x*) is used as the lhs side of an assignment, as in **x = [1, 2]*. This is not a SyntaxError on Python 3 though.
- Detect a couple of objects which can't be base classes (bool, slice, range and memoryview, which weren't detected until now).
- Add a new error for the Python 3 porting checker, *import-star-module-level*, which is used when a star import is detected in another scope than the module level, which is an error on Python 3. Using this will emit a SyntaxWarning on Python 2.

- Add a new error, 'star-needs-assignment-target', emitted on Python 3 when a Starred expression (*x) is not used in an assignment target. This is not caught when parsing the AST on Python 3, so it needs to be a separate check.
- Add a new error, 'unsupported-binary-operation', emitted when two a binary arithmetic operation is executed between two objects which don't support it (a number plus a string for instance). This is currently disabled, since the it exhibits way too many false positives, but it will be reenabled as soon as possible.
- New imported features from astroid into pyreverse: pyreverse.inspector.Project, pyreverse.inspector.project_from_files and pyreverse.inspector.interfaces.

  These were moved since they didn't belong in astroid.
- Enable misplaced-future for Python 3. Closes issue #580.
- Add a new error, 'nonlocal-and-global', which is emitted when a name is found to be both nonlocal and global in the same scope. Closes issue #581.
- ignored-classes option can work with qualified names (ignored-classes=optparse.Values) Closes issue #297.
- ignored-modules can work with qualified names as well as with Unix pattern matching for recursive ignoring. Closes issues #244.
- Improve detection of relative imports in non-packages, as well as importing missing modules with a relative import from a package.
- Don't emit no-init if not all the bases from a class are known. Closes issue #604.
- –no-space-check option accepts *empty-line* as a possible option. Closes issue #541.
- –generate-rcfile generates by default human readable symbols for the –disable option. Closes issue #608.
- Improved the not-in-loop checker to properly detect more cases.
- Add a new error, 'continue-in-finally', which is emitted when the *continue* keyword is found inside a *finally* clause, which is a SyntaxError.
- The –zope flag is deprecated and it is slated for removal in Pylint 1.6.

  The reason behind this removal is the fact that it's a specialized flag and there are solutions for the original problem: use –generated-members with the members that causes problems when using Zope or add AST transforms tailored to the zope project.

  At the same time, –include-ids and –symbols will also be removed in Pylint 1.6. Closes issue #570.
- missing-module-attribute was removed and the corresponding CLI option, required-attributes, which is slated for removal in Pylint 1.6.
- missing-reversed-argument was removed.

  The reason behind this is that this kind of errors should be detected by the type checker for *all* the builtins and not as a special case for the reversed builtin. This will happen shortly in the future.
- –comment flag is obsolete and it will be removed in Pylint 1.6.
- –profile flag is obsolete and it will be removed in Pylint 1.6.
- Add a new error, 'misplaced-bare-raise'.

  The error is used when a bare raise is not used inside an except clause. This can generate a RuntimeError in Python, if there are no active exceptions to be reraised. While it works in Python 2 due to the fact that the exception leaks outside of the except block, it's nevertheless a behaviour that a user shouldn't depend upon, since it's not obvious to the reader of the code what exception will be raised and it will not be compatible with Python 3 anyhow. Closes issue #633.

- Bring logilab-common's ureports into pylint.reporters.

  With this change, we moved away from depending on logilab-common, having in Pylint all the components that were used from logilab-common. The API should be considered an implementation detail and can change at some point in the future. Closes issue #621.

- *reimported* is emitted for reimported objects on the same line.

  Closes issue #639.

- Abbreviations of command line options are not supported anymore.

  Using abbreviations for CLI options was never considered to be a feature of pylint, this fact being only a side effect of using optparse. As this was the case, using –load-plugin or other abbreviation for –load-plugins never actually worked, while it also didn't raise an error. Closes issue #424.

- Add a new error, 'nonlocal-without-binding'

  The error is emitted on Python 3 when a nonlocal name is not bound to any variable in the parents scopes. Closes issue #582.

- **'deprecated-module' can be shown for modules which aren't** available. Closes issue #362.

- Don't consider a class abstract if its members can't be properly inferred.

  This fixes a false positive related to abstract-class-instantiated. Closes issue #648.

- Add a new checker for the async features added by PEP 492.

- Add a new error, 'yield-inside-async-function', emitted on Python 3.5 and upwards when the *yield* statement is found inside a new coroutine function (PEP 492).

- Add a new error, 'not-async-context-manager', emitted when an async context manager block is used with an object which doesn't support this protocol (PEP 492).

- Add a new convention warning, 'singleton-comparison', emitted when comparison to True, False or None is found.

- Don't emit 'assigning-non-slot' for descriptors. Closes issue #652.

- Add a new error, 'repeated-keyword', when a keyword argument is passed multiple times into a function call.

  This is similar with redundant-keyword-arg, but it's mildly different that it needs to be a separate error.

- –enable=all can now be used. Closes issue #142.

- Add a new convention message, 'misplaced-comparison-constant', emitted when a constant is placed in the left hand side of a comparison, as in '5 == func()'. This is also called Yoda condition, since the flow of code reminds of the Star Wars green character, conditions usually encountered in languages with variabile assignments in conditional statements.

- Add a new convention message, 'consider-using-enumerate', which is emitted when code that uses *range* and *len* for iterating is encountered. Closes issue #684.

- Added two new refactoring messages, 'no-classmethod-decorator' and 'no-staticmethod-decorator', which are emitted when a static method or a class method is declared without using decorators syntax.

  Closes issue #675.

9.5.24 What's New in Pylint 1.4.3?

Release date: 2015-03-14

- Remove three warnings: star-args, abstract-class-little-used, abstract-class-not-used. These warnings don't add any real value and they don't imply errors or problems in the code.

- Added a new option for controlling the peephole optimizer in astroid. The option `--optimize-ast` will control the peephole optimizer, which is used to optimize a couple of AST subtrees. The current problem solved by the peephole optimizer is when multiple joined strings, with the addition operator, are encountered. If the numbers of such strings is high enough, Pylint will then fail with a maximum recursion depth exceeded error, due to its visitor architecture. The peephole just transforms such calls, if it can, into the final resulting string and this exhibit a problem, because the visit_binop method stops being called (in the optimized AST it will be a Const node).

9.5.25 What's New in Pylint 1.4.2?

Release date: 2015-03-11

- Don't require a docstring for empty modules. Closes issue #261.
- Fix a false positive with *too-few-format-args* string warning, emitted when the string format contained a normal positional argument ('{0}'), mixed with a positional argument which did an attribute access ('{0.__class__}'). Closes issue #463.
- Take in account all the methods from the ancestors when checking for too-few-public-methods. Closes issue #471.
- Catch enchant errors and emit 'invalid-characters-in-docstring' when checking for spelling errors. Closes issue #469.
- Use all the inferred statements for the super-init-not-called check. Closes issue #389.
- Add a new warning, 'unichr-builtin', emitted by the Python 3 porting checker, when the unichr builtin is found. Closes issue #472.
- Add a new warning, 'intern-builtin', emitted by the Python 3 porting checker, when the intern builtin is found. Closes issue #473.
- Add support for editable installations.
- The HTML output accepts the *–msg-template* option. Patch by Dan Goldsmith.
- Add 'map-builtin-not-iterating' (replacing 'implicit-map-evaluation'), 'zip-builtin-not-iterating', 'range-builtin-not-iterating', and 'filter-builtin-not-iterating' which are emitted by *–py3k* when the appropriate built-in is not used in an iterating context (semantics taken from 2to3).
- Add a new warning, 'unidiomatic-typecheck', emitted when an explicit typecheck uses type() instead of isinstance(). For example, *type(x) == Y* instead of *isinstance(x, Y)*. Patch by Chris Rebert. Closes issue #299.
- Add support for combining the Python 3 checker mode with the –jobs flag (–py3k and –jobs). Closes issue #467.
- Add a new warning for the Python 3 porting checker, 'using-cmp-argument', emitted when the *cmp* argument for the *list.sort* or *sorted builtin* is encountered.
- Make the –py3k flag commutative with the -E flag. Also, this patch fixes the leaks of error messages from the Python 3 checker when the errors mode was activated. Closes issue #437.

9.5.26 What's New in Pylint 1.4.1?

Release date: 2015-01-16

- Look only in the current function's scope for bad-super-call. Closes issue #403.
- Check the return of properties when checking for not-callable. Closes issue #406.
- Warn about using the input() or round() built-ins for Python 3. Closes issue #411.

- Proper abstract method lookup while checking for abstract-class-instantiated. Closes issue #401.
- Use a mro traversal for finding abstract methods. Closes issue #415.
- Fix a false positive with catching-non-exception and tuples of exceptions.
- Fix a false negative with raising-non-exception, when the raise used an uninferrable exception context.
- Fix a false positive on Python 2 for raising-bad-type, when raising tuples in the form 'raise (ZeroDivisionError, None)'.
- Fix a false positive with invalid-slots-objects, where the slot entry was a unicode string on Python 2. Closes issue #421.
- Add a new warning, 'redundant-unittest-assert', emitted when using unittest's methods assertTrue and assertFalse with constant value as argument. Patch by Vlad Temian.
- Add a new JSON reporter, usable through -f flag.
- Add the method names for the 'signature-differs' and 'argument-differs' warnings. Closes issue #433.
- Don't compile test files when installing.
- Fix a crash which occurred when using multiple jobs and the files given as argument didn't exist at all.

9.5.27 What's New in Pylint 1.4.0?

Release date: 2014-11-23

- Added new options for controlling the loading of C extensions. By default, only C extensions from the stdlib will be loaded into the active Python interpreter for inspection, because they can run arbitrary code on import. The option *–extension-pkg-whitelist* can be used to specify modules or packages that are safe to load.
- Change default max-line-length to 100 rather than 80
- Drop BaseRawChecker class which were only there for backward compat for a while now
- Don't try to analyze string formatting with objects coming from function arguments. Closes issue #373.
- Port source code to be Python 2/3 compatible. This drops the need for 2to3, but does drop support for Python 2.5.
- Each message now comes with a confidence level attached, and can be filtered base on this level. This allows to filter out all messages that were emitted even though an inference failure happened during checking.
- Improved presenting unused-import message. Closes issue #293.
- Add new checker for finding spelling errors. New messages: wrong-spelling-in-comment, wrong-spelling-in-docstring. New options: spelling-dict, spelling-ignore-words.
- Add new '-j' option for running checks in sub-processes.
- Added new checks for line endings if they are mixed (LF vs CRLF) or if they are not as expected. New messages: mixed-line-endings, unexpected-line-ending-format. New option: expected-line-ending-format.
- 'dangerous-default-value' no longer evaluates the value of the arguments, which could result in long error messages or sensitive data being leaked. Closes issue #282
- Fix a false positive with string formatting checker, when encountering a string which uses only position-based arguments. Closes issue #285.
- Fix a false positive with string formatting checker, when using keyword argument packing. Closes issue #288.
- Proper handle class level scope for lambdas.

- Handle 'too-few-format-args' or 'too-many-format-args' for format strings with both named and positional fields. Closes issue #286.
- Analyze only strings by the string format checker. Closes issue #287.
- Properly handle nested format string fields. Closes issue #294.
- Don't emit 'attribute-defined-outside-init' if the attribute was set by a function call in a defining method. Closes issue #192.
- Properly handle unicode format strings for Python 2. Closes issue #296.
- Don't emit 'import-error' if an import was protected by a try-except, which excepted ImportError.
- Fix an 'unused-import' false positive, when the error was emitted for all the members imported with 'from import' form. Closes issue #304.
- Don't emit 'invalid-name' when assigning a name in an ImportError handler. Closes issue #302.
- Don't count branches from nested functions.
- Fix a false positive with 'too-few-format-args', when the format strings contains duplicate manual position arguments. Closes issue #310.
- fixme regex handles comments without spaces after the hash. Closes issue #311.
- Don't emit 'unused-import' when a special object is imported (__all__, __doc__ etc.). Closes issue #309.
- Look in the metaclass, if defined, for members not found in the current class. Closes issue #306.
- Don't emit 'protected-access' if the attribute is accessed using a property defined at the class level.
- Detect calls of the parent's __init__, through a binded super() call.
- Check that a class has an explicitly defined metaclass before emitting 'old-style-class' for Python 2.
- Emit 'catching-non-exception' for non-class nodes. Closes issue #303.
- Order of reporting is consistent.
- Add a new warning, 'boolean-datetime', emitted when an instance of 'datetime.time' is used in a boolean context. Closes issue #239.
- Fix a crash which ocurred while checking for 'method-hidden', when the parent frame was something different than a function.
- Generate html output for missing files. Closes issue #320.
- Fix a false positive with 'too-many-format-args', when the format string contains mixed attribute access arguments and manual fields. Closes issue #322.
- Extend the cases where 'undefined-variable' and 'used-before-assignment' can be detected. Closes issue #291.
- Add support for customising callback identifiers, by adding a new '–callbacks' command line option. Closes issue #326.
- Add a new warning, 'logging-format-interpolation', emitted when .format() string interpolation is used within logging function calls.
- Don't emit 'unbalanced-tuple-unpacking' when the rhs of the assignment is a variable length argument. Closes issue #329.
- Add a new warning, 'inherit-non-class', emitted when a class inherits from something which is not a class. Closes issue #331.
- Fix another false positives with 'undefined-variable', where the variable can be found as a class assignment and used in a function annotation. Closes issue #342.

- Handle assignment of the string format method to a variable. Closes issue #351.
- Support wheel packaging format for PyPi. Closes issue #334.
- Check that various built-ins that do not exist in Python 3 are not used: apply, basestring, buffer, cmp, coerce, execfile, file, long raw_input, reduce, StandardError, unicode, reload and xrange.
- Warn for magic methods which are not used in any way in Python 3: __coerce__, __delslice__, __getslice__, __setslice__, __cmp__, __oct__, __nonzero__ and __hex__.
- Don't emit 'assigning-non-slot' when the assignment is for a property. Closes issue #359.
- Fix for regression: '{path}' was no longer accepted in '–msg-template'.
- Report the percentage of all messages, not just for errors and warnings. Closes issue #319.
- 'too-many-public-methods' is reported only for methods defined in a class, not in its ancestors. Closes issue #248.
- 'too-many-lines' disable pragma can be located on any line, not only the first. Closes issue #321.
- Warn in Python 2 when an import statement is found without a corresponding *from __future__ import absolute_import*.
- Warn in Python 2 when a non-floor division operation is found without a corresponding *from __future__ import division*.
- Add a new option, 'exclude-protected', for excluding members from the protected-access warning. Closes issue #48.
- Warn in Python 2 when using dict.iter*(), dict.view*(); none of these methods are available in Python 3.
- Warn in Python 2 when calling an object's next() method; Python 3 uses __next__() instead.
- Warn when assigning to __metaclass__ at a class scope; in Python 3 a metaclass is specified as an argument to the 'class' statement.
- Warn when performing parameter tuple unpacking; it is not supported in Python 3.
- 'abstract-class-instantiated' is also emitted for Python 2. It was previously disabled.
- Add 'long-suffix' error, emitted when encountering the long suffix on numbers.
- Add support for disabling a checker, by specifying an 'enabled' attribute on the checker class.
- Add a new CLI option, –py3k, for enabling Python 3 porting mode. This mode will disable all other checkers and will emit warnings and errors for constructs which are invalid or removed in Python 3.
- Add 'old-octal-literal' to Python 3 porting checker, emitted when encountering octals with the old syntax.
- Add 'implicit-map-evaluation' to Python 3 porting checker, emitted when encountering the use of map builtin, without explicit evaluation.

9.5.28 What's New in Pylint 1.3.0?

Release date: 2014-07-26

- Allow hanging continued indentation for implicitly concatenated strings. Closes issue #232.
- Pylint works under Python 2.5 again, and its test suite passes.
- Fix some false positives for the cellvar-from-loop warnings. Closes issue #233.
- Return new astroid class nodes when the inferencer can detect that that result of a function invocation on a type (like *type* or *abc.ABCMeta*) is requested. Closes #205.

- Emit 'undefined-variable' for undefined names when using the Python 3 *metaclass*= argument.
- Checkers respect priority now. Close issue #229.
- Fix a false positive regarding W0511. Closes issue #149.
- Fix unused-import false positive with Python 3 metaclasses (#143).
- Don't warn with 'bad-format-character' when encountering the 'a' format on Python 3.
- Add multiple checks for PEP 3101 advanced string formatting: 'bad-format-string', 'missing-format-argument-key', 'unused-format-string-argument', 'format-combined-specification', 'missing-format-attribute' and 'invalid-format-index'.
- Issue broad-except and bare-except even if the number of except handlers is different than 1. Fixes issue #113.
- Issue attribute-defined-outside-init for all cases, not just for the last assignment. Closes issue #262.
- Emit 'not-callable' when calling properties. Closes issue #268.
- Fix a false positive with unbalanced iterable unpacking, when encountering starred nodes. Closes issue #273.
- Add new checks, 'invalid-slice-index' and 'invalid-sequence-index' for invalid sequence and slice indices.
- Add 'assigning-non-slot' warning, which detects assignments to attributes not defined in slots.
- Don't emit 'no-name-in-module' for ignored modules. Closes issue #223.
- Fix an 'unused-variable' false positive, where the variable is assigned through an import. Closes issue #196.
- Definition order is considered for classes, function arguments and annotations. Closes issue #257.
- Don't emit 'unused-variable' when assigning to a nonlocal. Closes issue #275.
- Do not let ImportError propagate from the import checker, leading to crash in some namespace package related cases. Closes issue #203.
- Don't emit 'pointless-string-statement' for attribute docstrings. Closes issue #193.
- Use the proper mode for pickle when opening and writing the stats file. Closes issue #148.
- Don't emit hidden-method message when the attribute has been monkey-patched, you're on your own when you do that.
- Only emit attribute-defined-outside-init for definition within the same module as the offended class, avoiding to mangle the output in some cases.
- Don't emit 'unnecessary-lambda' if the body of the lambda call contains call chaining. Closes issue #243.
- Don't emit 'missing-docstring' when the actual docstring uses *.format*. Closes issue #281.

9.5.29 What's New in Pylint 1.2.1?

Release date: 2014-04-30

- Restore the ability to specify the init-hook option via the configuration file, which was accidentally broken in 1.2.0.
- Add a new warning [bad-continuation] for badly indentend continued lines.
- Emit [assignment-from-none] when the function contains bare returns. Fixes BitBucket issue #191.
- Added a new warning for closing over variables that are defined in loops. Fixes Bitbucket issue #176.
- Do not warn about u escapes in string literals when Unicode literals are used for Python 2.*. Fixes BitBucket issue #151.

- Extend the checking for unbalanced-tuple-unpacking and unpacking-non-sequence to instance attribute unpacking as well.
- Fix explicit checking of python script (1.2 regression, #219)
- Restore –init-hook, renamed accidentally into –init-hooks in 1.2.0 (#211)
- Add 'indexing-exception' warning, which detects that indexing an exception occurs in Python 2 (behaviour removed in Python 3).

9.5.30 What's New in Pylint 1.2.0?

Release date: 2014-04-18

- Pass the current python paths to pylint process when invoked via epylint. Fixes BitBucket issue #133.
- Add -i / –include-ids and -s / –symbols back as completely ignored options. Fixes BitBucket issue #180.
- Extend the number of cases in which logging calls are detected. Fixes bitbucket issue #182.
- Improve pragma handling to not detect pylint:* strings in non-comments. Fixes BitBucket issue #79.
- Do not crash with UnknownMessage if an unknown message ID/name appears in disable or enable in the configuration. Patch by Cole Robinson. Fixes bitbucket issue #170.
- Add new warning 'eval-used', checking that the builtin function *eval* was used.
- Make it possible to show a naming hint for invalid name by setting include-naming-hint. Also make the naming hints configurable. Fixes BitBucket issue #138.
- Added support for enforcing multiple, but consistent name styles for different name types inside a single module; based on a patch written by morbo@google.com.
- Also warn about empty docstrings on overridden methods; contributed by sebastianu@google.com.
- Also inspect arguments to constructor calls, and emit relevant warnings; contributed by sebastianu@google.com.
- Added a new configuration option logging-modules to make the list of module names that can be checked for 'logging-not-lazy' et. al. configurable; contributed by morbo@google.com.
- ensure init-hooks is evaluated before other options, notably load-plugins (#166)
- Python 2.5 support restored: fixed small issues preventing pylint to run on python 2.5. Bitbucket issues #50 and #62.
- bitbucket #128: pylint doesn't crash when looking for used-before-assignment in context manager assignments.
- Add new warning, 'bad-reversed-sequence', for checking that the reversed() builtin receive a sequence (implements __getitem__ and __len__, without being a dict or a dict subclass) or an instance which implements __reversed__.
- Mark *file* as a bad function when using python2 (closes #8).
- Add new warning 'bad-exception-context', checking that *raise ... from ...* uses a proper exception context (None or an exception).
- Enhance the check for 'used-before-assignment' to look for 'nonlocal' uses.
- Emit 'undefined-all-variable' if a package's __all__ variable contains a missing submodule (closes #126).
- Add a new warning 'abstract-class-instantiated' for checking that abstract classes created with *abc* module and with abstract methods are instantied.
- Do not warn about 'return-arg-in-generator' in Python 3.3+.
- Do not warn about 'abstract-method' when the abstract method is implemented through assignment (#155).

- Improve cyclic import detection in the case of packages, patch by Buck Golemon
- Add new warnings for checking proper class __slots__: *invalid-slots-object* and *invalid-slots*.
- Search for rc file in *~/.config/pylintrc* if *~/.pylintrc* doesn't exists (#121)
- Don't register the newstyle checker w/ python >= 3
- Fix unused-import false positive w/ augment assignment (#78)
- Fix access-member-before-definition false negative wrt aug assign (#164)
- Do not attempt to analyze non python file, eg .so file (#122)

9.5.31 What's New in Pylint 1.1.0?

Release date: 2013-12-22

- Add new check for use of deprecated pragma directives "pylint:disable-msg" or "pylint:enable-msg" (I0022, deprecated-pragma) which was previously emmited as a regular warn().
- Avoid false used-before-assignment for except handler defined identifier used on the same line (#111).
- Combine 'no-space-after-operator', 'no-space-after-comma' and 'no-space-before-operator' into a new warning 'bad-whitespace'.
- Add a new warning 'superfluous-parens' for unnecessary parentheses after certain keywords.
- Fix a potential crash in the redefine-in-handler warning if the redefined name is a nested getattr node.
- Add a new option for the multi-statement warning to allow single-line if statements.
- Add 'bad-context-manager' error, checking that '__exit__' special method accepts the right number of arguments.
- Run pylint as a python module 'python -m pylint' (anatoly techtonik).
- Check for non-exception classes inside an except clause.
- epylint support options to give to pylint after the file to analyze and have basic input validation (bitbucket #53 and #54), patches provided by felipeochoa and Brian Lane.
- Added a new warning, 'non-iterator-returned', for non-iterators returned by '__iter__'.
- Add new checks for unpacking non-sequences in assignments (unpacking-non-sequence) as well as unbalanced tuple unpacking (unbalanced-tuple-unpacking).
- useless-else-on-loop not emited if there is a break in the else clause of inner loop (#117).
- don't mark *input* as a bad function when using python3 (#110).
- badly-implemented-container caused several problems in its current implementation. Deactivate it until we have something better. See #112 for instance.
- Use attribute regexp for properties in python3, as in python2
- Create the PYLINTHOME directory when needed, it might fail and lead to spurious warnings on import of pylint.config.
- Fix setup.py so that pylint properly install on Windows when using python3
- Various documentation fixes and enhancements
- Fix issue #55 (false-positive trailing-whitespace on Windows)

9.5.32 What's New in Pylint 1.0.0?

Release date: 2013-08-06

- Add check for the use of 'exec' function
- New –msg-template option to control output, deprecating "msvc" and "parseable" output formats as well as killing *–include-ids* and *–symbols* options
- Do not emit [fixme] for every line if the config value 'notes' is empty, but [fixme] is enabled.
- Emit warnings about lines exceeding the column limit when those lines are inside multiline docstrings.
- Do not double-check parameter names with the regex for parameters and inline variables.
- Added a new warning missing-final-newline (C0304) for files missing the final newline.
- Methods that are decorated as properties are now treated as attributes for the purposes of name checking.
- Names of derived instance class member are not checked any more.
- Names in global statements are now checked against the regular expression for constants.
- For toplevel name assignment, the class name regex will be used if pylint can detect that value on the right-hand side is a class (like collections.namedtuple()).
- Simplified invalid-name message
- Added a new warning invalid-encoded-data (W0512) for files that contain data that cannot be decoded with the specified or default encoding.
- New warning bad-open-mode (W1501) for calls to open (or file) that specify invalid open modes (Original implementation by Sasha Issayev).
- New warning old-style-class (C1001) for classes that do not have any base class.
- Add new name type 'class_attribute' for attributes defined in class scope. By default, allow both const and variable names.
- New warning trailing-whitespace (C0303) that warns about trailing whitespace.
- Added a new warning unpacking-in-except (W0712) about unpacking exceptions in handlers, which is unsupported in Python 3.
- Add a configuration option for missing-docstring to optionally exempt short functions/methods/classes from the check.
- Add the type of the offending node to missing-docstring and empty-docstring.
- New utility classes for per-checker unittests in testutils.py
- Do not warn about redefinitions of variables that match the dummy regex.
- Do not treat all variables starting with _ as dummy variables, only _ itself.
- Make the line-too-long warning configurable by adding a regex for lines for with the length limit should not be enforced
- Do not warn about a long line if a pylint disable option brings it above the length limit
- Do not flag names in nested with statements as undefined.
- Added a new warning 'old-raise-syntax' for the deprecated syntax raise Exception, args
- Support for PEP 3102 and new missing-kwoa (E1125) message for missing mandatory keyword argument (logilab.org's #107788)
- Fix spelling of max-branchs option, now max-branches

- Added a new base class and interface for checkers that work on the tokens rather than the syntax, and only tokenize the input file once.
- Follow astng renaming to astroid
- bitbucket #37: check for unbalanced unpacking in assignments
- bitbucket #25: fix incomplete-protocol false positive for read-only containers like tuple
- bitbucket #16: fix False positive E1003 on Python 3 for argument-less super()
- bitbucket #6: put back documentation in source distribution
- bitbucket #15: epylint shouldn't hang anymore when there is a large output on pylint'stderr
- bitbucket #7: fix epylint w/ python3
- bitbucket #3: remove string module from the default list of deprecated modules

9.5.33 What's New in Pylint 0.28.0?

Release date: 2013-04-25

- bitbucket #1: fix "dictionary changed size during iteration" crash
- #74013: new E1310[bad-str-strip-call] message warning when a call to a {l,r,}strip method contains duplicate characters (patch by Torsten Marek)
- #123233: new E0108[duplicate-argument-name] message reporting duplicate argument names
- #81378: emit W0120[useless-else-on-loop] for loops without break
- #124660: internal dependencies should not appear in external dependencies report
- #124662: fix name error causing crash when symbols are included in output messages
- #123285: apply pragmas for warnings attached to lines to physical source code lines
- #123259: do not emit E0105 for yield expressions inside lambdas
- #123892: don't crash when attempting to show source code line that can't be encoded with the current locale settings
- Simplify checks for dangerous default values by unifying tests for all different mutable compound literals.
- Improve the description for E1124[redundant-keyword-arg]

9.5.34 What's New in Pylint 0.27.0?

Release date: 2013-02-26

- #20693: replace pylint.el by Ian Eure version (patch by J.Kotta)
- #105327: add support for –disable=all option and deprecate the 'disable-all' inline directive in favour of 'skip-file' (patch by A.Fayolle)
- #110840: add messages I0020 and I0021 for reporting of suppressed messages and useless suppression pragmas. (patch by Torsten Marek)
- #112728: add warning E0604 for non-string objects in __all__ (patch by Torsten Marek)
- #120657: add warning W0110/deprecated-lambda when a map/filter of a lambda could be a comprehension (patch by Martin Pool)

- #113231: logging checker now looks at instances of Logger classes in addition to the base logging module. (patch by Mike Bryant)
- #111799: don't warn about octal escape sequence, but warn about o which is not octal in Python (patch by Martin Pool)
- #110839: bind <F5> to Run button in pylint-gui
- #115580: fix erroneous W0212 (access to protected member) on super call (patch by Martin Pool)
- #110853: fix a crash when an __init__ method in a base class has been created by assignment rather than direct function definition (patch by Torsten Marek)
- #110838: fix pylint-gui crash when include-ids is activated (patch by Omega Weapon)
- #112667: fix emission of reimport warnings for mixed imports and extend the testcase (patch by Torsten Marek)
- #112698: fix crash related to non-inferable __all__ attributes and invalid __all__ contents (patch by Torsten Marek)
- Python 3 related fixes:
- **#110213: fix import of checkers broken with python 3.3, causing** "No such message id W0704" breakage
- #120635: redefine cmp function used in pylint.reporters
- Include full warning id for I0020 and I0021 and make sure to flush warnings after each module, not at the end of the pylint run. (patch by Torsten Marek)
- Changed the regular expression for inline options so that it must be preceeded by a # (patch by Torsten Marek)
- Make dot output for import graph predictable and not depend on ordering of strings in hashes. (patch by Torsten Marek)
- Add hooks for import path setup and move pylint's sys.path modifications into them. (patch by Torsten Marek)

9.5.35 What's New in Pylint 0.26.0?

Release date: 2012-10-05

- #106534: add –ignore-imports option to code similarity checking and 'symilar' command line tool (patch by Ry4an Brase)
- #104571: check for anomalous backslash escape, introducing new W1401 and W1402 messages (patch by Martin Pool)
- #100707: check for boolop being used as exception class, introducing new W0711 message (patch by Tim Hatch)
- #4014: improve checking of metaclass methods first args, introducing new C0204 message (patch by lothiraldan@gmail.com finalized by sthenault)
- #4685: check for consistency of a module's __all__ variable, introducing new E0603 message
- #105337: allow custom reporter in output-format (patch by Kevin Jing Qiu)
- #104420: check for protocol completness and avoid false R0903 (patch by Peter Hammond)
- #100654: fix grammatical error for W0332 message (using 'l' as long int identifier)
- #103656: fix W0231 false positive for missing call to object.__init__ (patch by lothiraldan@gmail.com)
- #63424: fix similarity report disabling by properly renaming it to RP0801
- #103949: create a console_scripts entry point to be used by easy_install, buildout and pip

- fix cross-interpreter issue (non compatible access to __builtins__)
- stop including tests files in distribution, they causes crash when installed with python3 (#72022, #82417, #76910)

9.5.36 What's New in Pylint 0.25.2?

Release date: 2012-07-17

- #93591: Correctly emit warnings about clobbered variable names when an except handler contains a tuple of names instead of a single name. (patch by tmarek@google.com)
- #7394: W0212 (access to protected member) not emited on assigments (patch by lothiraldan@gmail.com)
- #18772; no prototype consistency check for mangled methods (patch by lothiraldan@gmail.com)
- #92911: emit W0102 when sets are used as default arguments in functions (patch by tmarek@google.com)
- #77982: do not emit E0602 for loop variables of comprehensions used as argument values inside a decorator (patch by tmarek@google.com)
- #89092: don't emit E0202 (attribute hiding a method) on @property methods
- #92584: fix pylint-gui crash due to internal API change
- #87192: fix crash when decorators are accessed through more than one dot (for instance @a.b is fine, @a.b.c crash)
- #88914: fix parsing of –generated-members options, leading to crash when using a regexp value set
- fix potential crashes with utils.safe_infer raising InferenceError

9.5.37 What's New in Pylint 0.25.1?

Release date: 2011-12-08

- #81078: Warn if names in exception handlers clobber overwrite existing names (patch by tmarek@google.com)
- #81113: Fix W0702 messages appearing with the wrong line number. (patch by tmarek@google.com)
- #50461, #52020, #51222: Do not issue warnings when using 2.6's property.setter/deleter functionality (patch by dneil@google.com)
- #9188, #4024: Do not trigger W0631 if a loop variable is assigned in the else branch of a for loop.

9.5.38 What's New in Pylint 0.25.0?

Release date: 2011-10-7

- #74742: make allowed name for first argument of class method configurable (patch by Google)
- #74087: handle case where inference of a module return YES; this avoid some cases of "TypeError: '_Yes' object does not support indexing" (patch by Google)
- #74745: make "too general" exception names configurable (patch by Google)
- #74747: crash occurs when lookup up a special attribute in class scope (patch by google)
- #76920: crash if on eg "pylint –rcfile" (patch by Torsten Marek)
- #77237: warning for E0202 may be very misleading

- #73941: HTML report messages table is badly rendered

9.5.39 What's New in Pylint 0.24.0?

Release date: 2011-07-18

- #69738: add regular expressions support for "generated-members"
- ids of logging and string_format checkers have been changed: logging: 65 -> 12, string_format: 99 -> 13 Also add documentation to say that ids of range 1-50 shall be reserved to pylint internal checkers
- #69993: Additional string format checks for logging module: check for missing arguments, too many arguments, or invalid string formats in the logging checker module. Contributed by Daniel Arena
- #69220: add column offset to the reports. If you've a custom reporter, this change may break it has now location gain a new item giving the column offset.
- #60828: Fix false positive in reimport check
- #70495: absolute imports fail depending on module path (patch by Jacek Konieczny)
- #22273: Fix –ignore option documentation to match reality

9.5.40 What's New in Pylint 0.23.0?

Release date: 2011-01-11

- documentation update, add manpages
- several performance improvements
- finalize python3 support
- new W0106 warning 'Expression "%s" is assigned to nothing'
- drop E0501 and E0502 messages about wrong source encoding: not anymore interesting since it's a syntax error for python >= 2.5 and we now only support this python version and above.
- don't emit W0221 or W0222 when methods as variable arguments (eg *arg and/or **args). Patch submitted by Charles Duffy.

9.5.41 What's New in Pylint 0.22.0?

Release date: 2010-11-15

- python versions: minimal python3.x support; drop python < 2.5 support

9.5.42 What's New in Pylint 0.21.4?

Release date: 2010-10-27

- fix #48066: pylint crashes when redirecting output containing non-ascii characters
- fix #19799: "pylint -blah" exit with status 2
- update documentation

9.5.43 What's New in Pylint 0.21.3?

Release date: 2010-09-28

- restored python 2.3 compatibility. Along with logilab-astng 0.21.3 and logilab-common 0.52, this will much probably be the latest release supporting python < 2.5.

9.5.44 What's New in Pylint 0.21.2?

Release date: 2010-08-26

- fix #36193: import checker raise exception on cyclic import
- fix #28796: regression in –generated-members introduced pylint 0.20
- some documentation cleanups

9.5.45 What's New in Pylint 0.21.1?

Release date: 2010-06-04

- fix #28962: pylint crash with new options, due to missing stats data while writing the Statistics by types report
- updated man page to 0.21 or greater command line usage (fix debian #582494)

9.5.46 What's New in Pylint 0.21.0?

Release date: 2010-05-11

- command line updated (closes #9774, #9787, #9992, #22962):
- all enable-* / disable-* options have been merged into –enable / –disable
- BACKWARD INCOMPATIBLE CHANGE: short name of –errors-only becomes -E, -e being affected to –enable
- pylint –help output much simplified, with –long-help available to get the complete one
- revisited gui, thanks to students from Toronto university (they are great contributors to this release!)
- fix #21591: html reporter produces no output if reports is set to 'no'
- fix #4581: not Missing docstring (C0111) warning if a method is overridden
- fix #4683: Non-ASCII characters count double if utf8 encode
- fix #9018: when using defining-attr-method, method order matters
- fix #4595: Comma not followed by a space should not occurs on trailing comma in list/tuple/dict definition
- fix #22585: [Patch] fix man warnings for pyreverse.1 manpage
- fix #20067: AttributeError: 'NoneType' object has no attribute 'name' with with

9.5.47 What's New in Pylint 0.20.0?

Release date: 2010-03-01

- fix #19498: fix windows batch file

- fix #19339: pylint.el : non existing py-mod-map (closes Debian Bug report logs - #475939)
- implement #18860, new W0199 message on assert (a, b)
- implement #9776, 'W0150' break or return statement in finally block may swallow exception.
- fix #9263, __init__ and __new__ are checked for unused arguments
- fix #20991, class scope definitions ignored in a genexpr
- fix #5975, Abstract intermediate class not recognized as such
- fix #5977, yield and return statement have their own counters, no more R0911 (Too many return statements) when a function have many yield stamtements
- implement #5564, function / method arguments with leading "_" are ignored in arguments / local variables count.
- implement #9982, E0711 specific error message when raising NotImplemented
- remove –cache-size option

9.5.48 What's New in Pylint 0.19.0?

Release date: 2009-12-18

- implement #18947, #5561: checker for function arguments
- include James Lingard string format checker
- include simple message (ids) listing by Vincent Ferotin (#9791)
- –errors-only does not hide fatal error anymore
- include james Lingard patches for ++/– and duplicate key in dicts
- include James Lingard patches for function call arguments checker
- improved flymake code and doc provided by Derek Harland
- refactor and fix the imports checker
- fix #18862: E0601 false positive with lambda functions
- fix #8764: More than one statement on a single line false positive with try/except/finally
- fix #9215: false undefined variable error in lambda function
- fix for w0108 false positive (Nathaniel)
- fix test/fulltest.sh
- #5821 added a utility function to run pylint in another process (patch provide by Vincent Ferotin)

9.5.49 What's New in Pylint 0.18.0?

Release date: 2009-03-25

- tests ok with python 2.4, 2.5, 2.6. 2.3 not tested
- fix #8687, W0613 false positive on inner function
- fix #8350, C0322 false positive on multi-line string
- fix #8332: set E0501 line no to the first line where non ascii character has been found

- avoid some E0203 / E0602 false negatives by detecting respectively AttributeError / NameError
- implements #4037: don't issue W0142 (* or ** magic) when they are barely passed from /* arguments
- complete #5573: more complete list of special methods, also skip W0613 for python internal method
- don't show information messages by default
- integration of Yuen Ho Wong's patches on emacs lisp files

9.5.50 What's New in Pylint 0.17.0?

Release date: 2009-03-19

- semicolon check : move W0601 to W0301
- remove rpython : remove all rpython checker, modules and tests
- astng 0.18 compatibility: support for _ast module modifies interfaces

9.5.51 What's New in Pylint 0.16.0?

Release date: 2009-01-28

- change [enldis]able-msg-cat options: only accept message categories identified by their first letter (eg IRCWEF) without the need for comma as separator
- add epylint.bat script to fix Windows installation
- setuptools/easy_install support
- include a modified version of Maarten ter Huurne patch to avoid W0613 warning on arguments from overridden method
- implement #5575 drop dumb W0704 message) by adding W0704 to ignored messages by default
- new W0108 message, checking for suspicious lambda (provided by Nathaniel Manista)
- fix W0631, false positive reported by Paul Hachmann
- fix #6951: false positive with W0104
- fix #6949
- patches by Mads Kiilerich:
- implement #4691, make pylint exits with a non zero return status if any messages other then Information are issued
- fix #3711, #5626 (name resolution bug w/ decorator and class members)
- fix #6954

9.5.52 What's New in Pylint 0.15.2?

Release date: 2008-10-13

- fix #5672: W0706 weirdness (W0706 removed)
- fix #5998: documentation points to wrong url for mailing list
- fix #6022: no error message on wrong module names

- fix #6040: pytest doesn't run test/func_test.py

9.5.53 What's New in Pylint 0.15.1?

Release date: 2008-09-15

- fix #4910: default values are missing in manpage
- fix #5991: missing files in 0.15.0 tarball
- fix #5993: epylint should work with python 2.3

9.5.54 What's New in Pylint 0.15.0?

Release date: 2008-09-10

- include pyreverse package and class diagram generation
- included Stefan Rank's patch to deal with 2.4 relative import
- included Robert Kirkpatrick's tutorial and typos fixes
- fix bug in reenabling message
- fix #2473: invoking pylint on __init__.py (hopefully)
- typecheck: acquired-members option has been dropped in favor of the more generic generated-members option. If the zope option is set, the behaviour is now to add some default values to generated-members.
- flymake integration: added bin/epylint and elisp/pylint-flymake.el

9.5.55 What's New in Pylint 0.14.0?

Release date: 2008-01-14

- fix #3733: Messages (dis)appear depending on order of file names
- fix #4026: pylint.el should require compile
- fix a bug in colorized reporter, spotted by Dave Borowitz
- applied patch from Stefan Rank to avoid W0410 false positive when multiple "from __future__" import statements
- implement #4012: flag back tick as deprecated (new W0333 message)
- new ignored-class option on typecheck checker allowing to skip members checking based on class name (patch provided by Thomas W Barr)

9.5.56 What's New in Pylint 0.13.2?

Release date: 2007-06-07

- fix disable-checker option so that it won't accidentally enable the rpython checker which is disabled by default
- added note about the gedit plugin into documentation

9.5.57 What's New in Pylint 0.13.1?

Release date: 2007-03-02

- fix some unexplained 0.13.0 packaging issue which led to a bunch of files missing from the distribution

9.5.58 What's New in Pylint 0.13.0?

Release date: 2007-02-28

- new RPython (Restricted Python) checker for PyPy fellow or people wanting to get a compiled version of their python program using the translator of the PyPy project. For more information about PyPy or RPython, visit http://codespeak.net/pypy/
- new E0104 and E0105 messages introduced to respectively warn about "return" and "yield" outside function or method
- new E0106 message when "yield" and "return something" are mixed in a function or method
- new W0107 message for unnecessary pass statement
- new W0614 message to differentiate between unused *import X* and unused *from X import ** (#3209, patch submitted by Daniel Drake)
- included Daniel Drake's patch to have a different message E1003 instead of E1001 when a missing member is found but an inference failure has been detected
- msvs reporter for Visual Studio line number reporting (#3285)
- allow disable-all option inline (#3218, patch submitted by Daniel Drake)
- –init-hook option to call arbitrary code necessary to set environment (eg sys.path) (#3156)
- One more Daniel's patch fixing a command line option parsing problem, this'll definitely be the DDrake release :)
- fix #3184: crashes on "return" outside function
- fix #3205: W0704 false positive
- fix #3123: W0212 false positive on static method
- fix #2485: W0222 false positive
- fix #3259: when a message is explicitly enabled, check the checker emitting it is enabled

9.5.59 What's New in Pylint 0.12.2?

Release date: 2006-11-23

- fix #3143: W0233 bug w/ YES objects
- fix #3119: Off-by-one error counting lines in a file
- fix #3117: ease sys.stdout overriding for reporters
- fix #2508: E0601 false positive with lambda
- fix #3125: E1101 false positive and a message duplication. Only the last part is actually fixed since the initial false positive is due to dynamic setting of attributes on the decimal.Context class.
- fix #3149: E0101 false positives and introduced E0100 for generator __init__ methods
- fixed some format checker false positives

9.5.60 What's New in Pylint 0.12.1?

Release date: 2006-09-25

- fixed python >= 2.4 format false positive with multiple lines statement
- fixed some 2.5 issues
- fixed generator expression scope bug (depends on astng 0.16.1)
- stop requiring __revision__

9.5.61 What's New in Pylint 0.12.0?

Release date: 2006-08-10

- usability changes:
  - parseable, html and color options are now handled by a single output-format option
  - enable-<checkerid> and disable-all options are now handled by two (exclusive) enable-checker and disable-checker options taking a comma separated list of checker names as value
  - renamed debug-mode option to errors-only
- started a reference user manual
- new W0212 message for access to protected member from client code (close #14081)
- new W0105 and W0106 messages extracted from W0104 (statement seems to have no effect) respectively when the statement is actually string (that's sometimes used instead of comments for documentation) or an empty statement generated by a useless semicolon
- reclassified W0302 to C0302
- fix so that global messages are not anymore connected to the last analyzed module (close #10106)
- fix some bugs related to local disabling of messages
- fix cr/lf pb when generating the rc file on windows platforms

9.5.62 What's New in Pylint 0.11.0?

Release date: 2006-04-19

- fix crash caused by the exceptions checker in some case
- fix some E1101 false positive with abstract method or classes defining __getattr__
- dirty fix to avoid "_socketobject" has not "connect" member. The actual problem is that astng isn't able to understand the code used to create socket.socket object with exec
- added an option in the similarity checker to ignore docstrings, enabled by default
- included patch from Benjamin Niemann to allow block level enabling/disabling of messages

9.5.63 What's New in Pylint 0.10.0?

Release date: 2006-03-06

- WARNING, this release include some configuration changes (see below), so you may have to check and update your own configuration file(s) if you use one

- this release require the 0.15 version of astng or superior (it will save you a lot of pylint crashes...)
- W0705 has been reclassified to E0701, and is now detecting more inheriting problem, and a false positive when empty except clause is following an Exception catch has been fixed (close #10422)
- E0212 and E0214 (metaclass/class method should have mcs/cls as first argument have been reclassified to C0202 and C0203 since this not as well established as "self" for instance method (E0213)
- W0224 has been reclassified into F0220 (failed to resolve interfaces implemented by a class)
- a new typecheck checker, introducing the following checks:
  - E1101, access to unexistent member (implements #10430), remove the need of E0201 and so some options has been moved from the classes checker to this one
  - E1102, calling a non callable object
  - E1111 and W1111 when an assignment is done on a function call but the inferred function returns None (implements #10431)
- change in the base checker:
  - checks module level and instance attribute names (new const-rgx and attr-rgx configuration option) (implements #10209 and #10440)
  - list comprehension and generator expression variables have their own regular expression (the inlinevar-rgx option) (implements #9146)
  - the C0101 check with its min-name-length option has been removed (this can be specified in the regxp after all...)
  - W0103 and W0121 are now handled by the variables checker (W0103 is now W0603 and W0604 has been splitted into different messages)
  - W0131 and W0132 messages have been reclassified to C0111 and C0112 respectively
  - new W0104 message on statement without effect
- regexp support for dummy-variables (dummy-variables-rgx option replace dummy-variables) (implements #10027)
- better global statement handling, see W0602, W0603, W0604 messages (implements #10344 and #10236)
- –debug-mode option, disabling all checkers without error message and filtering others to only display error
- fixed some R0201 (method could be a function) false positive

9.5.64 What's New in Pylint 0.9.0?

Release date: 2006-01-10

- a lot of updates to follow astng 0.14 API changes, so install logilab-astng 0.14 or greater before using this version of pylint
- checker number 10 ! newstyle will search for problems regarding old style / new style classes usage problems (rely on astng 0.14 new style detection feature)
- new 'load-plugins' options to load additional pylint plugins (usable from the command line or from a configuration file) (implements #10031)
- check if a "pylintrc" file exists in the current working directory before using the one specified in the PYLINTRC environment variable or the default ~/.pylintrc or /etc/pylintrc
- fixed W0706 (Identifier used to raise an exception is assigned...) false positive and reraising a catched exception instance

- fixed E0611 (No name get in module blabla) false positive when accessing to a class'__dict__
- fixed some E0203 ("access to member before its definition") false positive
- fixed E0214 ("metaclass method first argument should be mcs) false positive with staticmethod used on a metaclass
- fixed packaging which was missing the test/regrtest_data directory
- W0212 (method could be a function) has been reclassified in the REFACTOR category as R0201, and is no more considerer when a method overrides an abstract method from an ancestor class
- include module name in W0401 (wildcard import), as suggested by Amaury
- when using the '–parseable', path are written relative to the current working directory if in a sub-directory of it (#9789)
- 'pylint –version' shows logilab-astng and logilab-common versions
- fixed pylint.el to handle space in file names
- misc lint style fixes

9.5.65 What's New in Pylint 0.8.1?

Release date: 2005-11-07

- fix "deprecated module" false positive when the code imports a module whose name starts with a deprecated module's name (close #10061)
- fix "module has no name __dict__" false positive (close #10039)
- fix "access to undefined variable __path__" false positive (close #10065)
- fix "explicit return in __init__" false positive when return is actually in an inner function (close #10075)

9.5.66 What's New in Pylint 0.8.0?

Release date: 2005-10-21

- check names imported from a module exists in the module (E0611), patch contributed by Amaury Forgeot d'Arc
- print a warning (W0212) for methods that could be a function (implements #9100)
- new –defining-attr-methods option on classes checker
- new –acquired-members option on the classes checker, used when –zope=yes to avoid false positive on acquired attributes (listed using this new option) (close #8616)
- generate one E0602 for each use of an undefined variable (previously, only one for the first use but not for the following) (implements #1000)
- make profile option saveable
- fix Windows .bat file, patch contributed by Amaury Forgeot d'Arc
- fix one more false positive for E0601 (access before definition) with for loop such as "for i in range(10): print i" (test func_noerror_defined_and_used_on_same_line)
- fix false positive for E0201 (undefined member) when accessing to __name__ on a class object
- fix astng checkers traversal order
- fix bug in format checker when parsing a file from a platform using different new line characters (close #9239)

- fix encoding detection regexp
- fix –rcfile handling (support for –rcfile=file, close #9590)

9.5.67 What's New in Pylint 0.7.0?

Release date: 2005-05-27

- WARNING: pylint is no longer a logilab subpackage. Users may have to manually remove the old logilab/pylint directory.
- introduce a new –additional-builtins option to handle user defined builtins
- –reports option has now -r as short alias, and -i for –include-ids
- fix a bug in the variables checker which may causing some false positives when variables are defined and used within the same statement (test func_noerror_defined_and_used_on_same_line)
- this time, real fix of the "disable-msg in the config file" problem, test added to unittest_lint
- fix bug with –list-messages and python -OO
- fix possible false positive for W0201

9.5.68 What's New in Pylint 0.6.4?

Release date: 2005-04-14

- allow to parse files without extension when a path is given on the command line (test noext)
- don't fail if we are unable to read an inline option (e.g. inside a module), just produce an information message (test func_i0010)
- new message E0103 for break or continue outside loop (close #8883, test func_continue_not_in_loop)
- fix bug in the variables checker, causing non detection of some actual name error (close #8884, test func_nameerror_on_string_substitution)
- fix bug in the classes checker which was making pylint crash if "object" is assigned in a class inheriting from it (test func_noerror_object_as_class_attribute)
- fix problem with the similar checker when related options are defined in a configuration file
- new –generate-man option to generate pylint's man page (require the latest logilab.common (>= 0.9.3)
- packaged (generated. . .) man page

9.5.69 What's New in Pylint 0.6.3?

Release date: 2005-02-24

- fix scope problem which may cause false positive and true negative on E0602
- fix problem with some options such as disable-msg causing error when they are coming from the configuration file

9.5.70 What's New in Pylint 0.6.2?

Release date: 2005-02-16

- fix false positive on E0201 ("access to undefined member") with metaclasses
- fix false positive on E0203 ("access to member before its definition") when attributes are defined in a parent class
- fix false positive on W0706 ("identifier used to raise an exception assigned to...")
- fix interpretation of "t" as value for the indent-string configuration variable
- fix –rcfile so that –rcfile=pylintrc (only –rcfile pylintrc was working in earlier release)
- new raw checker example in the examples/ directory

9.5.71 What's New in Pylint 0.6.1?

Release date: 2005-02-04

- new –rcfile option to specify the configuration file without the PYLINTRC environment variable
- added an example module for a custom pylint checker (see the example/ directory)
- some fixes to handle fixes in common 0.9.1 (should however still working with common 0.9.0, even if upgrade is recommended)

9.5.72 What's New in Pylint 0.6.0?

Release date: 2005-01-20

- refix pylint emacs mode
- no more traceback when just typing "pylint"
- fix a bug which may cause crashes on resolving parent classes
- fix problems with the format checker: don't chock on files containing multiple CR, avoid C0322, C0323, C0324 false positives with triple quoted string with quote inside
- correctly detect access to member defined latter in __init__ method
- now depends on common 0.8.1 to fix problem with interface resolution (close #8606)
- new –list-msgs option describing available checkers and their messages
- added windows specific documentation to the README file, contributed by Brian van den Broek
- updated doc/features.txt (actually this file is now generated using the –list-msgs option), more entries into the FAQ
- improved tests coverage

9.5.73 What's New in Pylint 0.5.0?

Release date: 2004-10-19

- avoid to import analyzed modules !
- new Refactor and Convention message categories. Some Warnings have been remaped into those new categories

- added "similar", a tool to find copied and pasted lines of code, both using a specific command line tool and integrated as a pylint's checker
- imports checker may report import dependencies as a dot graph
- new checker regrouping most Refactor detection (with some new metrics)
- more command line options storable in the configuration file
- fix bug with total / undocumented number of methods

9.5.74 What's New in Pylint 0.4.2?

Release date: 2004-07-08

- fix pylint emacs mode
- fix classes checkers to handler twisted interfaces

9.5.75 What's New in Pylint 0.4.1?

Release date: 2004-05-14

- fix the setup.py script to allow bdist_winst (well, the generated installer has not been tested...) with the necessary logilab/__init__.py file
- fix file naming convention as suggested by Andreas Amoroso
- fix stupid crash bug with bad method names

9.5.76 What's New in Pylint 0.4.0?

Release date: 2004-05-10

- fix file path with –parsable
- –parsable option has been renamed to –parseable
- added patch from Andreas Amoroso to output message to files instead of standard output
- added Run to the list of correct variable names
- fix variable names regexp and checking of local classes names
- some basic handling of metaclasses
- no-docstring-rgx apply now on classes too
- new option to specify a different regexp for methods than for functions
- do not display the evaluation report when no statements has been analysed
- fixed crash with a class nested in a method
- fixed format checker to deals with triple quoted string and lines with code and comment mixed
- use logilab.common.ureports to layout reports

9.5.77 What's New in Pylint 0.3.3?

Release date: 2004-02-17

- added a parsable text output, used when the –parsable option is provided
- added an emacs mode using this output, available in the distrib's elisp directory
- fixed some typos in messages
- change include-ids options to yn, and allow it to be in the configuration file
- do not chock on corrupted stats files
- fixed bug in the format checker which may stop pylint execution
- provide scripts for unix and windows to wrap the minimal pylint tk gui

9.5.78 What's New in Pylint 0.3.2?

Release date: 2003-12-23

- html-escape messages in the HTML reporter (bug reported by Juergen Hermann)
- added "TODO" to the list of default note tags
- added "rexec" to the list of default deprecated modules
- fixed typos in some messages

9.5.79 What's New in Pylint 0.3.1?

Release date: 2003-12-05

- bug fix in format and classes checkers
- remove print statement from imports checkers
- provide a simple tk gui, essentially useful for windows users

9.5.80 What's New in Pylint 0.3.0?

Release date: 2003-11-20

- new exceptions checker, checking for string exception and empty except clauses.
- imports checker checks for reimport of modules
- classes checker checks for calls to ancestor's __init__ and abstract method not overridden. It doesn't complain anymore for unused import in __init__ files, and provides a new option ignore-interface-methods, useful when you're using zope Interface implementation in your project
- base checker checks for black listed builtins call (controled by the bad-functions option) and for use of * and **
- format checker checks for use of <> and "l" as long int marker
- major internal API changes
- use the rewrite of astng, based on compiler.ast
- added unique id for messages, as suggested by Wolfgang Grafen
- added unique id for reports

- can take multiple modules or files as argument
- new options command line options : –disable-msg, –enable-msg, –help-msg, –include-ids, –reports, –disable-report, –cache-size
- –version shows the version of the python interpreter
- removed some options which are now replaced by [en|dis]able-msg, or disable-report
- read disable-msg and enable-msg options in source files (should be in comments on the top of the file, in the form "# pylint: disable-msg=W0402"
- new message for modules importing themselves instead of the "cyclic import" message
- fix bug with relative and cyclic imports
- fix bug in imports checker (cycle was not always detected)
- still fixes in format checker : don't check comment and docstring, check first line after an indent
- black and white list now apply to all identifiers, not only variables, so changed the configuration option from (good|bad)-variable-names to (good|bad)-names
- added string, rexec and Bastion to the default list of deprecated modules
- do not print redefinition warning for function/class/method defined in mutually exclusive branches

9.5.81 What's New in Pylint 0.2.1?

Release date: 2003-10-10

- added some documentation, fixed some typos
- set environment variable PYLINT_IMPORT to 1 during pylint execution.
- check that variables "imported" using the global statement exist
- indentation problems are now warning instead of errors
- fix checkers.initialize to try to load all files with a known python extension (patch from wrobell)
- fix a bunch of messages
- fix sample configuration file
- fix the bad-construction option
- fix encoding checker
- fix format checker

9.5.82 What's New in Pylint 0.2.0?

Release date: 2003-09-12

- new source encoding / FIXME checker (pep 263)
- new –zope option which trigger Zope import. Useful to check Zope products code.
- new –comment option which enable the evaluation note comment (disabled by default).
- a ton of bug fixes
- easy functional test infrastructure

9.5.83 What's New in Pylint 0.1.2?

Release date: 2003-06-18

- bug fix release
- remove dependency to pyreverse

9.5.84 What's New in Pylint 0.1.1?

Release date: 2003-06-01

- much more functionalities !

9.5.85 What's New in Pylint 0.1?

Release date: 2003-05-19

- initial release

Index

Symbols

C