HOW TO WRITE PYLINT PLUGINS

Alexander Todorov
http://atodorov.org

https://github.com/PyCQA/pylint
https://github.com/PyCQA/pylint-django/

PiterPy 2018
Saint Petersburg

Rahul Gopinath
@_rahulgopinath

Following

Spent the day debugging (effectively) this silly thing in Python: ` v = (lambda: i*i for i in range(10));  [j() for j in list(v)]` -- remove `list` for fun!. For more, see rahul.gopinath.org /2018/09/10/whe ... #python

8:20 PM - 10 Sep 2018

3 Likes

3

Only if Rahul had this ?!?

```
class ListComprehensionChecker(checkers.BaseChecker):


    def visit_listcomp(self, node):
        self.add_message('avoid-list-comprehension',
                         node=node)
```

Example: Why write a plugin ?

```
$ server/api/
          .../cats.py
          .../dogs.py

  client/
     .../cats_consumer.py
     .../dogs_consumer.py
```

Parameter names, order & count must be the same

Example: Java-ism in Python

```
class Cat:
    def __init__(self, name):
        self._name = name

    def getName(self):
        return self._name

    def setName(self, new_name):
        self._name = new_name
```

OBJECTIVES

- To understand how a pylint plugin works
- To be able to write a non-complex plugin

INSTALLING AND RUNNING

```
$ pip install pylint
$ pylint *.py
```

RESULTS

```
$ pylint tcms_api/setup.py

No config file found, using default configuration

************* Module tcms_api.setup
C:  1, 0: Missing module docstring (missing-docstring)
C:  7, 0: Constant name "description" doesn't conform to
UPPER_CASE naming style (invalid-name)
C:  9, 4: Constant name "long_description" doesn't conform
to UPPER_CASE naming style (invalid-name)

-----------------------------------
Your code has been rated at 4.00/10
```

PLUGIN SKELETON

```
$ cat myplugin.py

def register(linter):
    print("Hello Pylint Plugins")
    #linter.register_checker(SomeCheckerClass(linter))
```

INVOKING THE PLUGIN

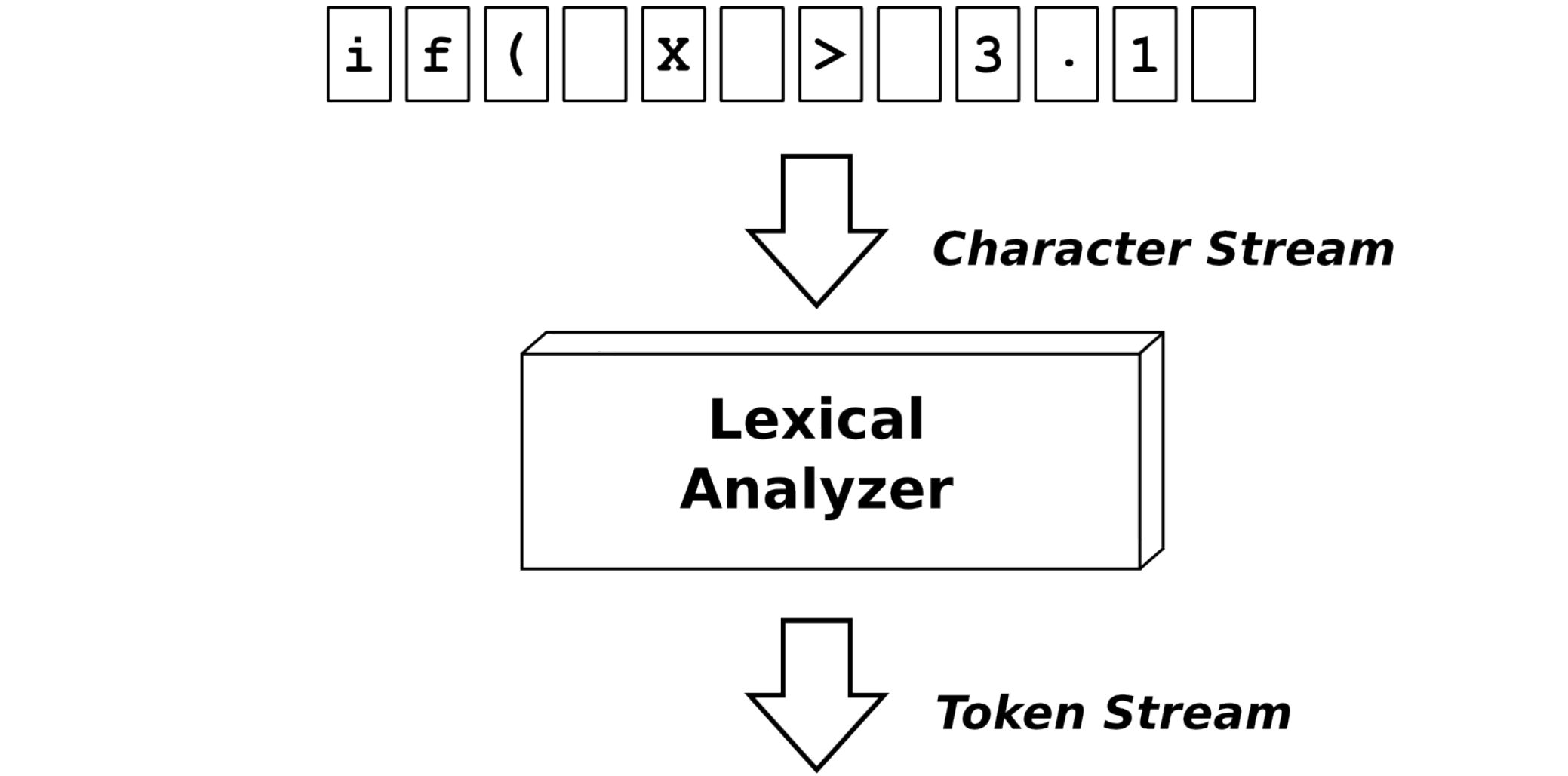
```
$ PYTHONPATH=. pylint --load-plugins=myplugin *.py

Hello Pylint Plugins

No config file found, using default configuration
************* Module myplugin
C:  1, 0: Missing module docstring (missing-docstring)
C:  2, 0: Missing function docstring (missing-docstring)
W:  2,13: Unused argument 'linter' (unused-argument)

-----------------------------------------------------------
Your code has been rated at -5.00/10 (previous run:
-5.00/10, +0.00)
```

PARSING & AST CRASH COURSE

```
while b ≠ 0
    if a > b
        a := a − b
    else
        b := b − a
return a
```

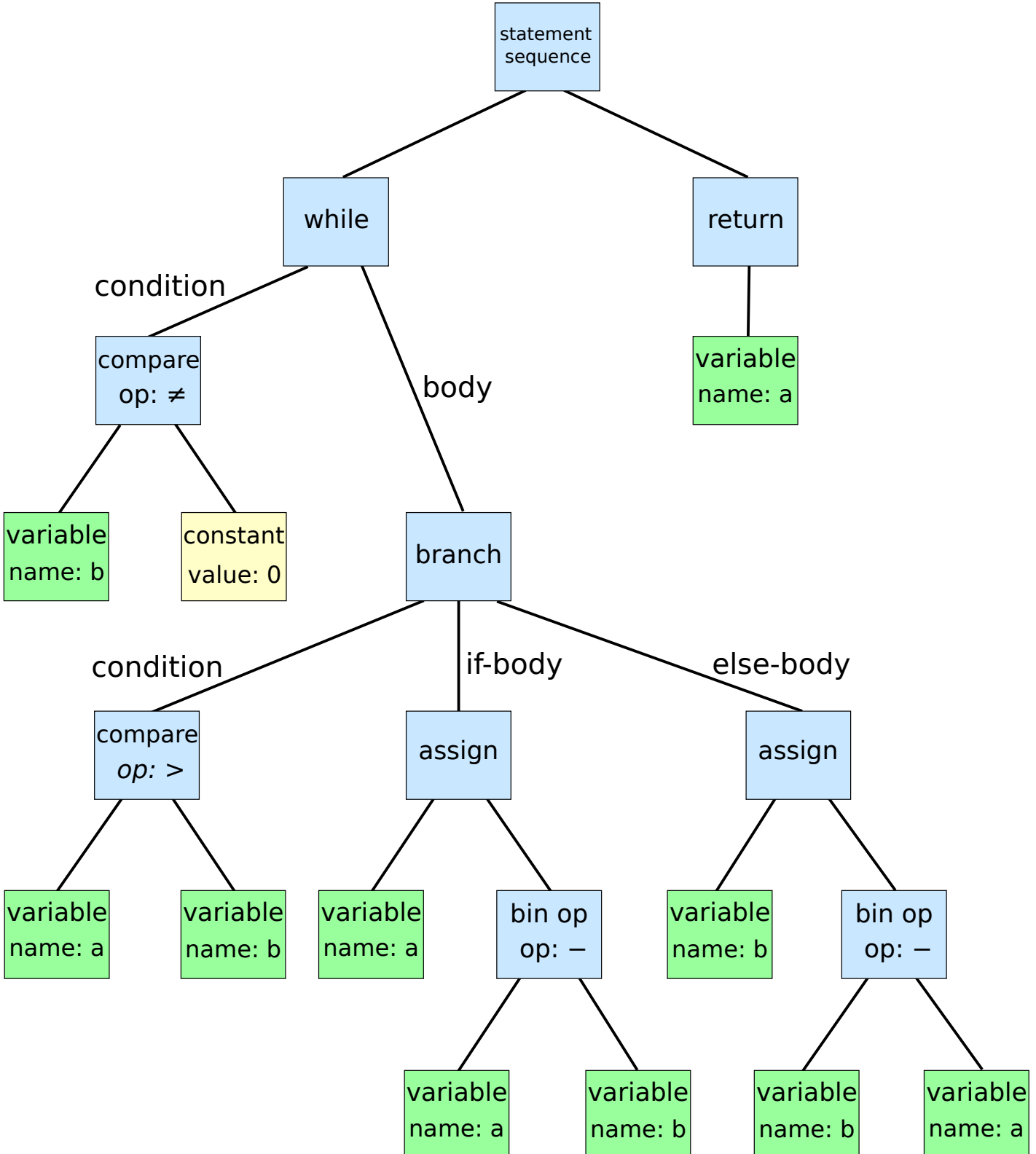
https://en.wikipedia.org/wiki/Parsing#Computer_languages

https://en.wikipedia.org/wiki/Abstract_syntax_tree


i f (x > 3 . 1
Character Stream
Lexical Analyzer
Token Stream
KEYWORD
"if"
BRACKET
"("
IDENTIFIER
"x"
OPERATOR
">"
NUMBER
"3.1"

TOKENIZING IN PYTHON

```
>>> import io, tokenize
>>> for token_info in tokenize.tokenize(io.BytesIO(
            b"""print('Hello World')""").readline):
...     print(token_info)

TokenInfo(type=59 (ENCODING), string='utf-8', start=(0, 0), end=(0, 0),
line='')

TokenInfo(type=1 (NAME), string='print', start=(1, 0), end=(1, 5),
line="print('Hello World')")

TokenInfo(type=53 (OP), string='(', start=(1, 5), end=(1, 6),
line="print('Hello World')")

TokenInfo(type=3 (STRING), string="'Hello World'", start=(1, 6),
end=(1, 19), line="print('Hello World')")

TokenInfo(type=53 (OP), string=')', start=(1, 19), end=(1, 20),
line="print('Hello World')")

TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0),
line='')
```


statement sequence
while
return
condition
compare op: ≠
body
variable name: a
variable name: b
constant value: 0
branch
condition
if-body
else-body
compare op: >
assign
assign
variable name: a
variable name: b
variable name: a
bin op op: −
variable name: b
bin op op: −
variable name: a
variable name: b
variable name: b
variable name: a

PARSING AST IN PYTHON

```
>>> import astroid

>>> print(astroid.parse('print("Hello World")').repr_tree())
Module(
   name='',
   doc=None,
   file='<?>',
   path=['<?>'],
   package=False,
   pure_python=True,
   future_imports=set(),
   body=[Expr(value=Call(
             func=Name(name='print'),
             args=[Const(value='Hello World')],
             keywords=None))])
```

AST CHEAT SHEET

- http://astroid.readthedocs.io/en/latest/api/astroid.nodes.html

Remember these two methods:

- `visit_<lowercase_node_class_name>(self, node)`
- `leave_<lowercase_node_class_name>(self, node)`

AST CHEAT SHEET

- http://astroid.readthedocs.io/en/latest/api/base_nodes.html

- `NodeNG.as_string(self)`
- `get_children()`
- `parent - the parent node in the syntax tree`

- `frame(self) - the first parent frame node(Module, FunctionDef, or ClassDef)`

CHECKER INTERFACES

```
IChecker:
  def open(self)
  def close(self)

IRawChecker(IChecker):
  def process_module(self, module)

ITokenChecker(IChecker):
  def process_tokens(self, tokens)

IAstroidChecker(IChecker):
  def visit_...(self, node)
  def leave_...(self, node)
```

CHECKER INTERFACES

```
IRawChecker(IChecker):
  def process_module(self, module)
```

```
module == astroid.parse(source_code_str)
```

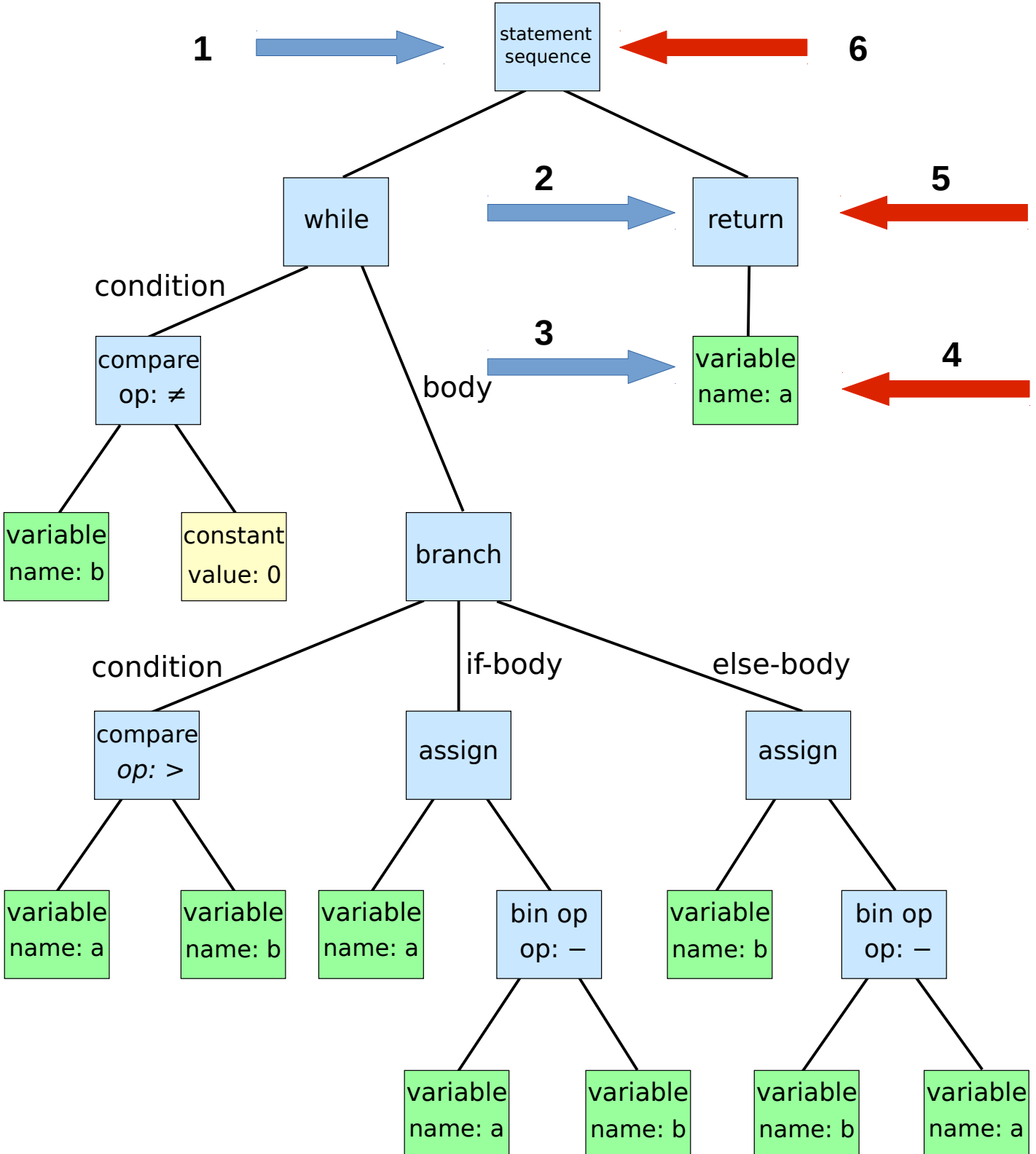
```
See:
pylint/checkers/mics.py::EncodingChecker
```

CHECKER INTERFACES

```
ITokenChecker(IChecker):
  def process_tokens(self, tokens)


tokens == tokenize.tokenize()
tokens == (token_type,
           token_value,
           (start_row, start_col),
           (end_row, end_col),
           line)
```

CHECKER INTERFACES

```
IAstroidChecker(IChecker):
  def visit_...(self, node)
  def leave_...(self, node)

... == lowercase astroid node class name
```


1
statement sequence
6
while
2
return
5
condition
compare op: ≠
3
variable name: a
4
body
variable name: b
constant value: 0
branch
condition
if-body
else-body
compare op: >
assign
assign
variable name: a
variable name: b
variable name: a
bin op op: −
variable name: b
bin op op: −
variable name: a
variable name: b
variable name: b
variable name: a

CHECKER SKELETON

```
from pylint import interfaces
from pylint import checkers


class AwesomeChecker(checkers.BaseChecker):
    __implements__ = (interfaces.IAstroidChecker, )

    name = 'awesome-checker'

    msgs = {'R9901': ('Short message',
                      'awesome-checker-message',
                      'Longer help message!')}

    def close(self):
        print('Awesome checker finished working')
        # self.add_message('awesome-checker-message',
        #                  node=None)

# don't forget linter.register_checker(AwesomeChecker(linter))
# inside myplugin.py::register()
```

MESSAGE HELP

```
$ pylint --list-msgs
$ pylint --help-msg <MESSAGE_ID | MESSAGE_CODE>
```

Kiwi TCMS

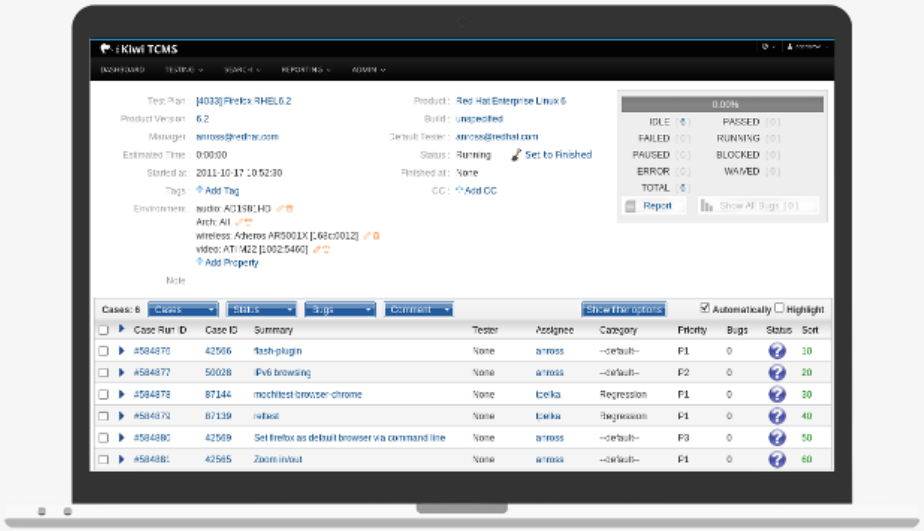
Checkout Demo | Documentation | Community | Support | Blog | Search

Kiwi TCMS

the leading open source test case management system

- Efficiently manage test cases, plans and runs
- Improve testing productivity & reporting
- Integrates with popular issue trackers
- External API interface
- GPL 2 licensed

Checkout Demo | See Features



Open source test case management system, with a lot of great features, such as bug tracker integration, fast search, powerful access control and external API.

Everyday testing

Use the dashboard to see pending work. Execute tests, mark results and report bugs. Collect automation results!

Test management

Define test plans and cases, track progress and assign work across multiple teams. Perform peer reviews.

Reporting

See who's doing what and provide status report to stakeholders. Centralize your acceptance books!

Integration

Integrated with Bugzilla, JIRA and GitHub The API interface provides full access so you can get creative!

Django 2 & Python 3

Patternfly & jQuery

Docker

Tested

https://github.com/kiwitcms/Kiwi/tree/master/kiwi_lint

AVOID LIST COMPREHENSIONS

```
from pylint import interfaces
from pylint import checkers


class ListComprehensionChecker(checkers.BaseChecker):
    __implements__ = (interfaces.IAstroidChecker,)

    name = 'list-comprehension-checker'

    msgs = {'R4411': ('Avoid using list comprehensions',
                      'avoid-list-comprehension',
                      'Just avoid them in Kiwi TCMS!')}

    def visit_listcomp(self, node):
        self.add_message('avoid-list-comprehension',
                         node=node)
```

USELESS RETURN

```
def add_cases(run_ids, case_ids):
    trs = TestRun.objects.filter(...)
    tcs = TestCase.objects.filter(..)

    for tr in trs.iterator():
        for tc in tcs.iterator():
            tr.add_case_run(case=tc)

    return
```

USELESS RETURN CHECKER

```
01  def visit_functiondef(self, node):
02      # if the function has empty body then exit
03      if not node.body:
04          return
05
06      last = node.body[-1]
07      if isinstance(last, astroid.Return):
08          if last.value is None:  # e.g. "return"
09              self.add_message('useless-return',
10                               node=node)
11          # e.g. "return None"
12          elif isinstance(last.value, astroid.Const) and
13               (last.value.value is None):
14              self.add_message('useless-return',
15                               node=node)
```

https://github.com/PyCQA/pylint/pull/1823

HARD-CODED auth.User

Instead of:

```
class Component(models.Model):
    name = models.CharField()
    initial_owner = models.ForeignKey('auth.User')
```

Django recommends:

```
class Component(models.Model):
    name = models.CharField()
    initial_owner = models.ForeignKey(
                                    settings.AUTH_USER_MODEL)
```

auth.User CHECKER

```
def visit_const(self, node):
    if node.value == 'auth.User':
        self.add_message('hard-coded-auth-user', node=node)

-----------------------------------------------------

def process_tokens(self, tokens):
    for (tok_type, token, (start_row, _), _, _) in tokens:
        if tok_type == tokenize.STRING and
           token.find('auth.User') > -1:
            self.add_message('hard-coded-auth-user',
                             line=start_row)
```

“““ for doc-strings

```
01      _string_tokens = {}
02
03      def process_tokens(self, tokens):
04          for (tok_type, token, _, _, _) in tokens:
05              if tok_type == tokenize.STRING:
06                  token_text = token.strip('"').strip("'")
07                  self._string_tokens[token_text] = token
08
09      def visit_{module|classdef|functiondef}(self, node):
10          self._check_docstring(node)
11
12      def _check_docstring(self, node):
13          if node.doc in self._string_tokens:
14              token = self._string_tokens[node.doc]
15              if not token.startswith('"""'):
16                  self.add_message(
                      'use-triple-double-quotes', node=node)
```

__dunder__ class attributes

```
class MyClass:
    __all__ = [„list“, „of“, „attributes“]
    __warnings__ = []

    ....
```

__dunder__ class attributes

```
01  def visit_classdef(self, node):
02      # e.g. __doc__, __iter__
03      allowed_attrs = dir(list)
04
05      for child in node.body:
06          if isinstance(child, astroid.Assign):
07              for target in child.targets:
08                  if (isinstance(target,
09                                 astroid.AssignName)   and
10                      target.name not in allowed_attrs and
11                      target.name.startswith('__')     and
12                      target.name.endswith('__')):
13                      self.add_message('dunder-class-attribute',
14                                       node=child)
```

QUICK HOWTO

- Find a pattern you want to check for
- Examine AST/tokens manually
- Find inspiration in pylint, pylint-django & Kiwi TCMS
- Create a bare-bones plugin/checker class
- Extend & refine checker class !!!

- *Transformation plugins – ping me afterwards or https://astroid.readthedocs.io/en/latest/extending.html*

THANK YOU !

Transformation plugins

- AST transformations
- Module extender transformations
- AST inference tips

http://astroid.readthedocs.io/en/latest/extending.html

AST transformations

```
astroid.MANAGER.register_transform(
    <astroid Node Type>,
    transform_function,
    predicate_function)
```

AST transformations example

```
01    class SomeModel(models.Model):
02        class Meta:
03            ordering = ('-id',)

Missing doc-string on line 02!
```

AST transformations example

```
def is_model_meta(node):
    if node.name != 'Meta' or
       not isinstance(node.parent, astroid.nodes.ClassDef):
        return False
    return True

def transform_model_meta(node):
    node.doc = 'auto generated documentation'
    return node

astroid.MANAGER.register_transform(
    astroid.nodes.ClassDef,
    transform_model_meta,
    is_model_meta)
```

AST transformations example

```
>>> import uuid
>>> uuid.uuid4()
UUID('a2b73af9-7621-4e2a-8a89-c7cc9d279668')
>>> uuid.uuid4().int
                ^^^ no-member
321776005601058774482884424498123589726


class UUID:
    def __init__(...., int=None, ...):
        ...
        self.__dict__[“int“] = int
```

AST transformations example

```
def _patch_uuid_class(node):
    node.locals['int'] = [nodes.Const(0, parent=node)]


astroid.MANAGER.register_transform(
    nodes.ClassDef,
    _patch_uuid_class,
    lambda node: node.qname() == 'uuid.UUID')
```

https://github.com/PyCQA/astroid/blob/master/astroid/brain/brain_uuid.py

Module extender example

```
>>> import curses
>>> curses.KEY_DOWN
258
>>> curses.__file__
'/opt/rh/rh-python36/root/usr/lib64/python3.6/curses/__init__.py'

__init__.py contains
from _curses import *

>>> _curses.__file__
'.../python3.6/lib-dynload/_curses.cpython-36m-x86_64-linux-gnu.so'
```

Module extender example

```
import astroid

def _curses_transform():
    return astroid.parse('''
    KEY_DOWN = 1''')

astroid.register_module_extender(
    astroid.MANAGER,
    'curses',
    _curses_transform)
```

https://github.com/PyCQA/astroid/blob/master/astroid/brain/brain_curses.py

Astroid inference

```
>>> name_node = astroid.extract_node('''
... a = 1
... b = 2
... c = a + b
... c
''')
>>> inferred = next(name_node.infer())
>>> inferred
<Const.int l.None at 0x10d913128>
>>> inferred.value
3
```

Inference tip example

```
from django.utils.translation import ugettext_lazy

ugettext_lazy('"{pk}" not found').format(pk=obj_pk)
                                  ^^^ no-member

Function definition:
ugettext_lazy = lazy(gettext, str)

django.utils.functional.lazy – proxy class inside
```

Inference tip example

```
def is_ugettext_lazy(node):
    return isinstance(node.func, astroid.nodes.Name) and
           node.func.name == 'ugettext_lazy'

def transform_ugettext_lazy(node, context=None):
    new_node = astroid.nodes.Const('', parent=node)
    return iter((new_node, )) # iterator of possible values

astroid.MANAGER.register_transform(
    astroid.nodes.Call,
    astroid.inference_tip(transform_ugettext_lazy),
    is_ugettext_lazy)
```