

# API Documentation

API Documentation

February 10, 2008

## Contents

<b>Contents</b>	<b>1</b>
<b>1 Module contract</b>	<b>2</b>
1.1 Functions . . . . .	2
1.2 Variables . . . . .	5
1.3 Class Done . . . . .	6
1.3.1 Methods . . . . .	6
1.4 Class tokenizer . . . . .	6
1.4.1 Methods . . . . .	6
1.5 Class ContractViolationError . . . . .	7
1.5.1 Methods . . . . .	7
1.6 Class PreconditionViolationError . . . . .	8
1.6.1 Methods . . . . .	8
1.7 Class PostconditionViolationError . . . . .	8
1.7.1 Methods . . . . .	8
1.8 Class InvariantViolationError . . . . .	9
1.8.1 Methods . . . . .	9
1.9 Class InvalidPreconditionError . . . . .	9
1.9.1 Methods . . . . .	9

# 1 Module contract

Programming-by-contract for Python, based on Eiffel’s DBC.

Programming by contract documents class and modules with invariants, expressions that must be true during the lifetime of a module or instance; and documents functions and methods with pre- and post-conditions that must be true during entry and return.

Copyright (c) 2003, 2006, 2007 Terence Way

This program is free software; you can redistribute it and/or modify it under the terms of either:

- a) GNU Library or Lesser General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version, or
- b) Python Software Foundation License You may redistribute and/or modify this program under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form, or
- c) The “Artistic License” which comes with this Kit.

You should have received a copy of the Artistic License with this Kit, in the file named “Artistic”. If not, I’ll be glad to provide one.

You should also have received a copy of the GNU Lesser General Public License along with this program in the files named “COPYING” and “COPYING.LESSER”. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston MA 02111-1307, USA or visit their web page on the internet at <http://www.gnu.org/copyleft/lgpl.html>.

IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS CODE, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CODE PROVIDED HEREUNDER IS ON AN “AS IS” BASIS, AND THERE IS NO OBLIGATION WHATSOEVER TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS. **Version:** 1.4: August 31, 2007

**Author:** Terence Way

**License:** Lesser GNU Public License, Python Software Foundation License, Artistic

## 1.1 Functions

<pre><b>checkmod</b>(<i>module</i>, <i>checklevel</i>=0)</pre>
<pre>Add invariant, pre- and post-condition checking to a module.</pre>
<pre>pre:</pre>
<pre>    isstring(<i>module</i>) or isinstance(<i>module</i>, ModuleType)</pre>
<pre>    <i>checklevel</i> in [CHECK_DEFAULT, CHECK_NONE, CHECK_PRECONDITIONS, CHECK_ALL]</pre>

**parse\_docstring**(*docstring*, *keywords*)

Parse a docstring, looking for design-by-contract expressions.

Returns a list of tuples: the list is the same length as keywords, and matches each keyword. The tuple is (keyword, [decls], [exprs]), namely the keyword, a list of string declarations, and a list of tuples (string, lineno).

Examples:

```
>>> from pprint import pprint
>>> pprint( parse_docstring(parse_docstring.__doc__, ['post', 'pre']) )
[('post', [], [(' x [ 0 ] for x in __return__ == keywords', 22)]),
 ('pre', [], [('docstring is None or isstring ( docstring )', 18),
 ('forall ( keywords , isstring )', 19)])]
```

pre:

```
docstring is None or isstring(docstring)
forall(keywords, isstring)
```

post[]:

```
[x[0] for x in __return__] == keywords
```

**isclass**(*obj*)**isstring**(*obj*)**getargspec**(*function*)

Get argument information about a function.

Returns a tuple (args, varargs, keywordvarargs, defaults) where args is a list of strings, varargs is None or the name of the \*va argument, keywordvarargs is None or the name of the \*\*ka argument, and defaults is a list of default values.

This function is different from the Python-provided inspect.getargspec in that 1) tuple arguments are returned as a string grouping '(a, b, c)' instead of broken out "[ 'a', 'b', 'c' ]" and 2) it works in Jython, which doesn't support inspect (yet).

```
>>> getargspec(lambda a, b: a * b)
(['a', 'b'], None, None, None)
>>> getargspec(lambda a, (b, c, d) = (5, 6, 7), *va, **ka: a * b)
(['a', '(b, c, d)', 'va', 'ka', ((5, 6, 7),))
```

pre:

```
isinstance(function, FunctionType)
```

post[]:

```
# tuple of form (args, va, ka, defaults)
isinstance(__return__, TupleType) and len(__return__) == 4
# args is a list of strings
isinstance(__return__[0], ListType)
forall(__return__[0], isstring)
# va is None or a string
__return__[1] is None or isstring(__return__[1])
# ka is None or a string
__return__[2] is None or isstring(__return__[2])
# defaults is None or a tuple
__return__[3] is None or isinstance(__return__[3], TupleType)
```

**call\_public\_function\_all**(*inv, func, \*va, \*\*ka*)

Check the invocation of a public function or static method.  
Checks module invariants on entry and exit. Checks any pre-conditions and post-conditions.

**call\_public\_function\_pre**(*inv, func, \*va, \*\*ka*)

Check the invocation of a public function or static method.  
Checks module invariants on entry. Checks any pre-conditions.

**call\_private\_function\_all**(*func, \*va, \*\*ka*)

Check the invocation of a private function or static method.  
Only checks pre-conditions and post-conditions.

**call\_private\_function\_pre**(*func, \*va, \*\*ka*)

Check the invocation of a private function or static method.  
Only checks pre-conditions

**call\_public\_method\_all**(*cls, method, \*va, \*\*ka*)

Check the invocation of a public method.  
Check this class and all super-classes invariants on entry and exit. Checks all post-conditions of this method and all over- ridden method.

**call\_public\_method\_pre**(*cls, method, \*va, \*\*ka*)

Check the invocation of a public method.  
Check this class and all super-classes invariants on entry. exit.

**call\_constructor\_all**(*cls, method, \*va, \*\*ka*)

Check the invocation of an `__init__` constructor.  
Checks pre-conditions and post-conditions, and only checks invariants on successful completion.

**call\_constructor\_pre**(*cls, method, \*va, \*\*ka*)

Check the invocation of an `__init__` constructor.  
Checks pre-conditions and post-conditions, and only checks invariants on successful completion.

**call\_destructor\_all**(*cls, method, \*va, \*\*ka*)

Check the invocation of a `__del__` destructor.  
Checks pre-conditions and post-conditions, and only checks invariants on entry.

**call\_destructor\_pre**(*cls, method, \*va, \*\*ka*)

Check the invocation of a `__del__` destructor.  
Checks pre-conditions and post-conditions, and only checks invariants on entry.

---

```
call_private_method_all(cls, method, *va, **ka)
```

---

Check the invocation of a private method call.  
Checks pre-conditions and post-conditions.

---

```
call_private_method_pre(cls, method, *va, **ka)
```

---

Check the invocation of a private method call.  
Checks pre-conditions.

---

```
forall(a, fn=<type 'bool'>)
```

---

Checks that all elements in a sequence are true.  
Returns True(1) if all elements are true. Return False(0) otherwise.

Examples:

```
>>> forall([True, True, True])
1
>>> forall( () )
1
>>> forall([True, True, False, True])
0
```

---

```
exists(a, fn=<type 'bool'>)
```

---

Checks that at least one element in a sequence is true.  
Returns True(1) if at least one element is true. Return False(0) otherwise.

Examples:

```
>>> exists([False, False, True])
1
>>> exists([])
0
>>> exists([False, 0, '', []])
0
```

---

```
implies(test, then_val, else_val=True)
```

---

Logical implication.

implies(x, y) should be read 'x implies y' or 'if x then y' implies(x, a, b) should be read 'if x then a else b'

Examples:

```
>>> implies(False, False)
1
>>> implies(False, True)
1
>>> implies(True, False)
0
>>> implies(True, True)
1
```

## 1.2 Variables

Name	Description
<code>__email__</code>	Value: 'terry@wayforward.net'
MODULE	Value: 'contract'
INV	Value: 'inv'
PRE	Value: 'pre'
POST	Value: 'post'
TYPE_CONTRACTS	Value: ['inv']
CODE_CONTRACTS	Value: ['pre', 'post']
OLD	Value: '__old__'
RETURN	Value: '__return__'
PREFIX	Value: '__assert__'
<code>__test__</code>	Value: {'_ispublic': _ispublic, '_get_members': _get_members, '...'
CHECK_ALL	Value: 3
CHECK_DEFAULT	Value: 0
CHECK_NONE	Value: 1
CHECK_PRECONDITIONS	Value: 2
CO_VARARGS	Value: 4
CO_VARKEYWORDS	Value: 8
False	Value: False
True	Value: True

### 1.3 Class Done

```

exceptions.Exception ┌
                    │
                    └─ contract.Done

```

#### 1.3.1 Methods

```
__getitem__(...)
```

```
__init__(...)
```

```
__str__(...)
```

### 1.4 Class tokenizer

#### 1.4.1 Methods

```
__init__(self, input, startlineno)
```

```
next(self, token, string, start, end, line)
```

```
start(self, token, string)
```

`indent(self, token, string)``name(self, token, string)``decl0(self, token, string)``decl1(self, token, string)``decln(self, token, string)``colon(self, token, string)``colon1(self, token, string)``colon2(self, token, string)``newline(self, token, string)``rest(self, token, string)`

## 1.5 Class `ContractViolationError`

exceptions.Exception └

exceptions.StandardError └

exceptions.AssertionError └

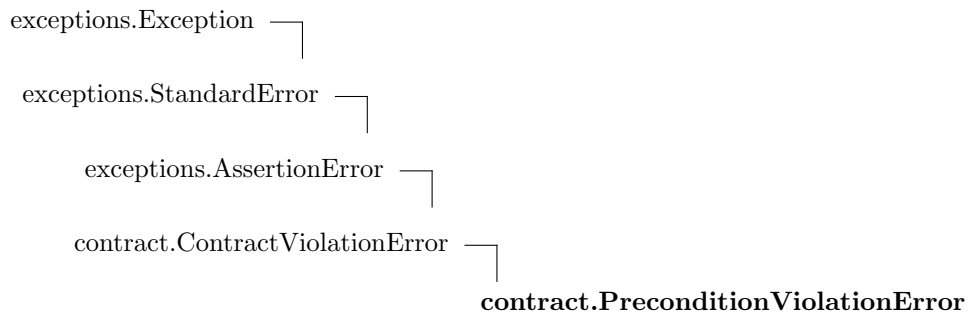
**contract.ContractViolationError**

**Known Subclasses:** `contract.InvalidPreconditionError`, `contract.InvariantViolationError`, `contract.PostconditionViolationError`, `contract.PreconditionViolationError`

### 1.5.1 Methods

`__getitem__(...)``__init__(...)``__str__(...)`

## 1.6 Class `PreconditionViolationError`



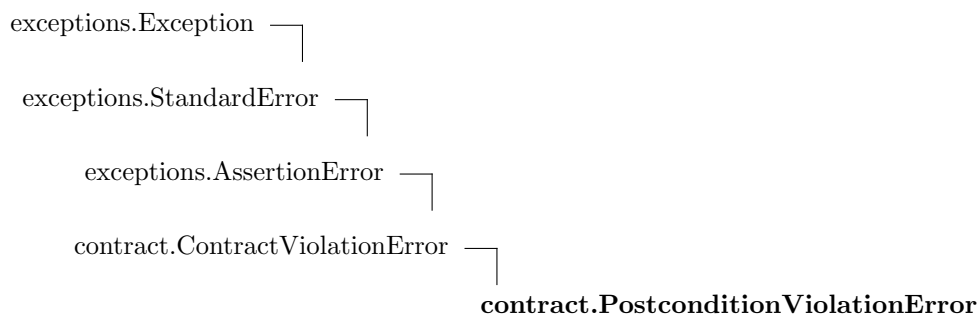
### 1.6.1 Methods

<code>__getitem__(...)</code>
-------------------------------

<code>__init__(...)</code>
----------------------------

<code>__str__(...)</code>
---------------------------

## 1.7 Class `PostconditionViolationError`



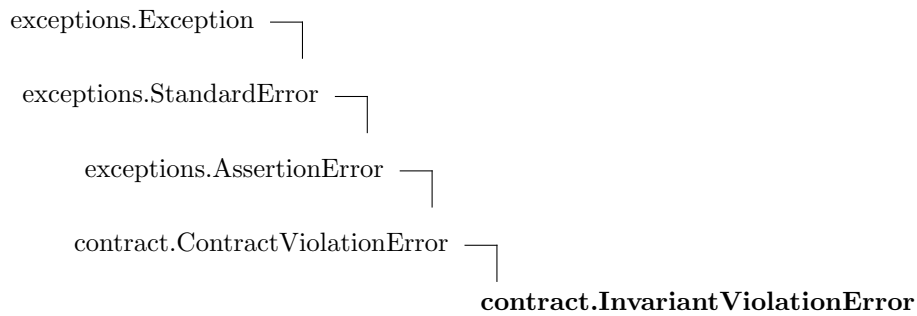
### 1.7.1 Methods

<code>__getitem__(...)</code>
-------------------------------

<code>__init__(...)</code>
----------------------------

<code>__str__(...)</code>
---------------------------

## 1.8 Class InvariantViolationError



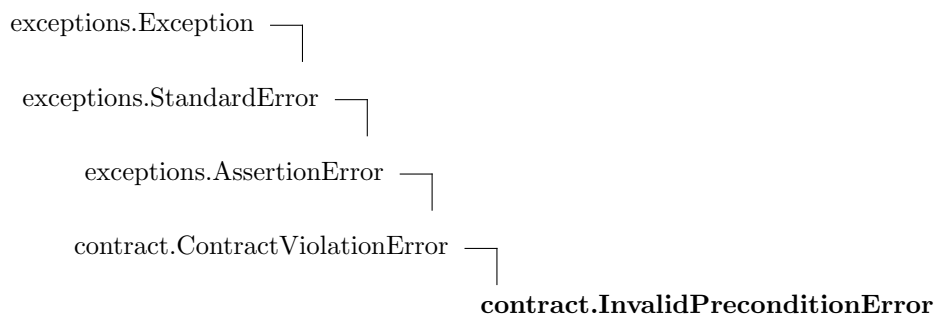
### 1.8.1 Methods

<code>__getitem__(...)</code>
-------------------------------

<code>__init__(...)</code>
----------------------------

<code>__str__(...)</code>
---------------------------

## 1.9 Class InvalidPreconditionError



Method pre-conditions can only weaken overridden methods' preconditions.

### 1.9.1 Methods

<code>__getitem__(...)</code>
-------------------------------

<code>__init__(...)</code>
----------------------------

<code>__str__(...)</code>
---------------------------