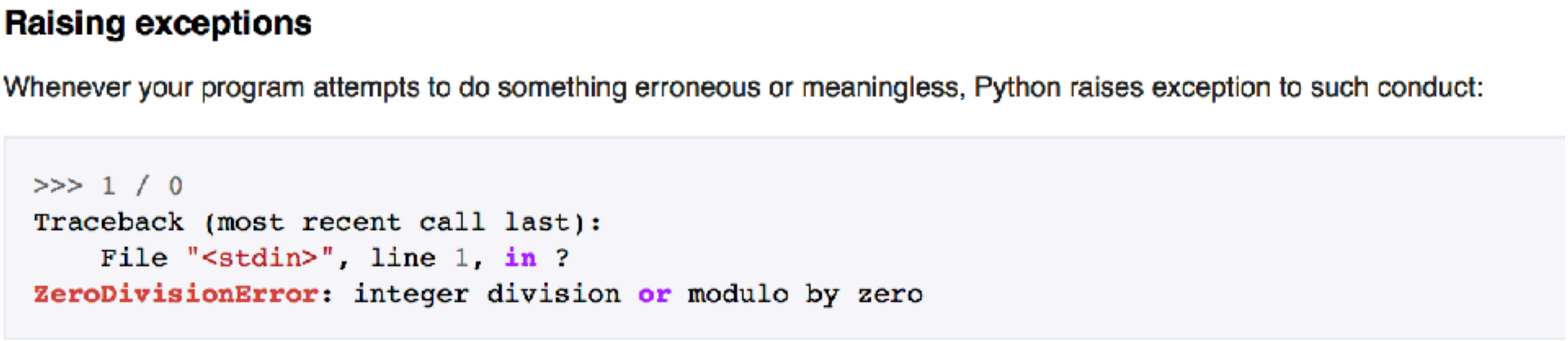
Try and Error

Python debugging
and
beautification

What happens when something goes wrong



Raising exceptions

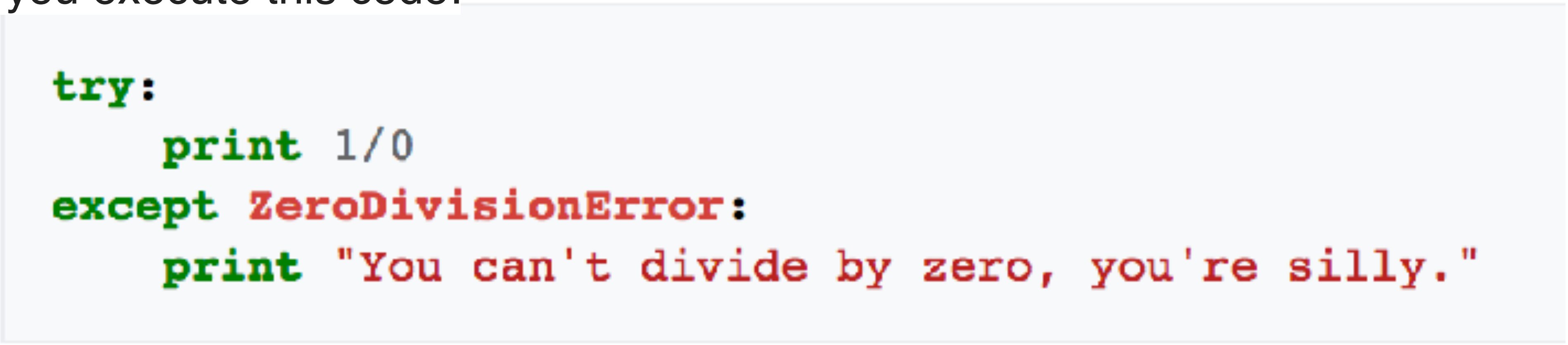
Whenever your program attempts to do something erroneous or meaningless, Python raises exception to such conduct:

```
>>> 1 / 0
Traceback (most recent call last):
    File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

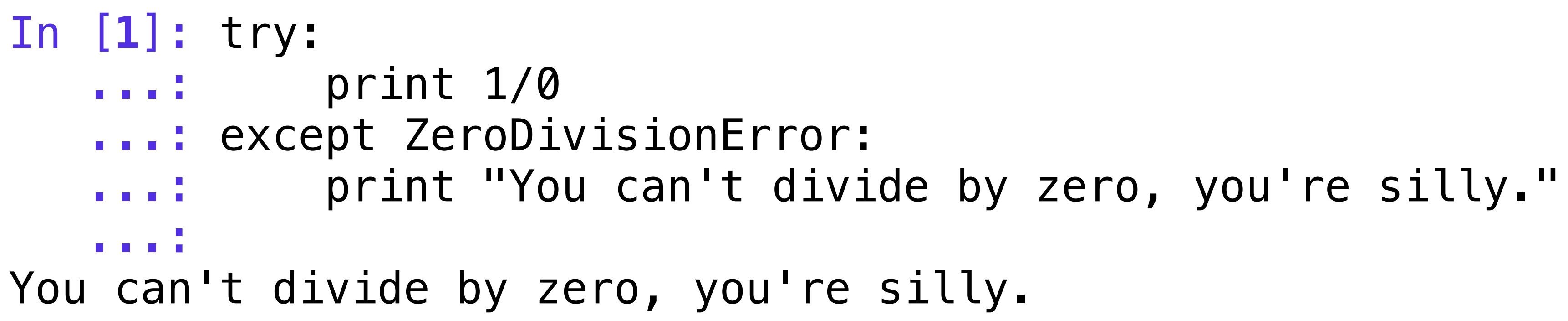
Catching exceptions

In order to handle errors, you can set up *exception handling blocks* in your code. The keywords `try` and `except` are used to catch exceptions. When an error occurs within the `try` block, Python looks for a matching `except` block to handle it. If there is one, execution jumps there.

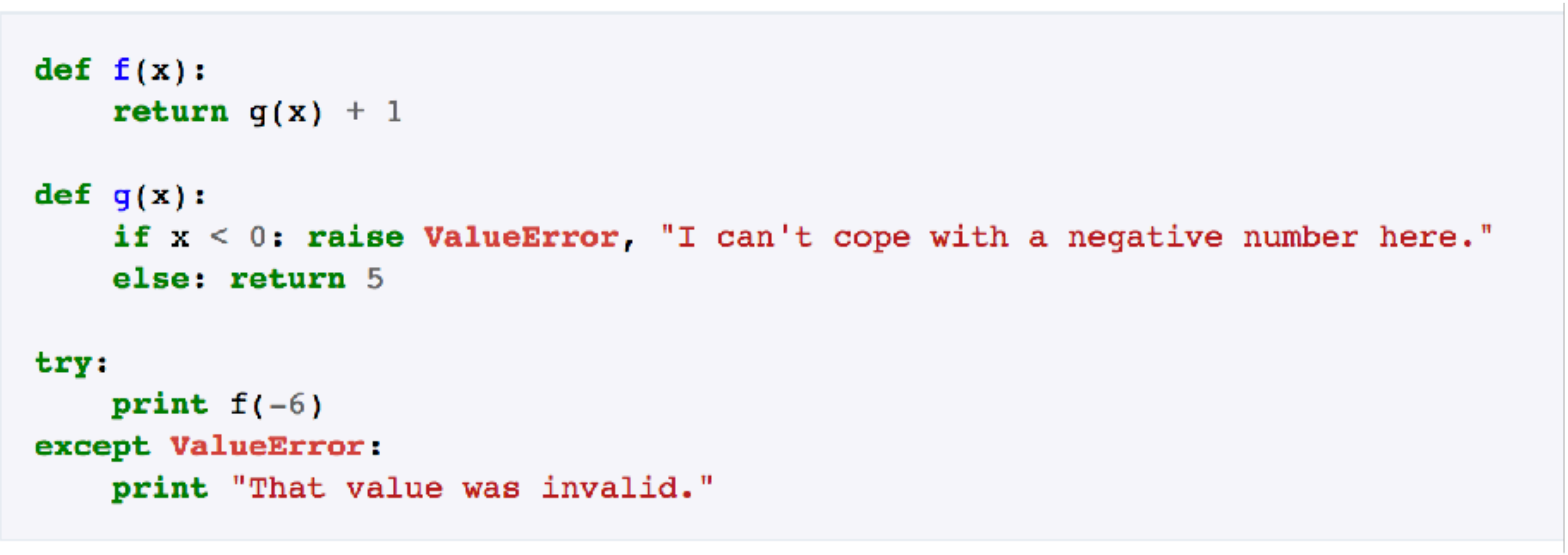
If you execute this code:



```
try:
    print 1/0
except ZeroDivisionError:
    print "You can't divide by zero, you're silly."
```



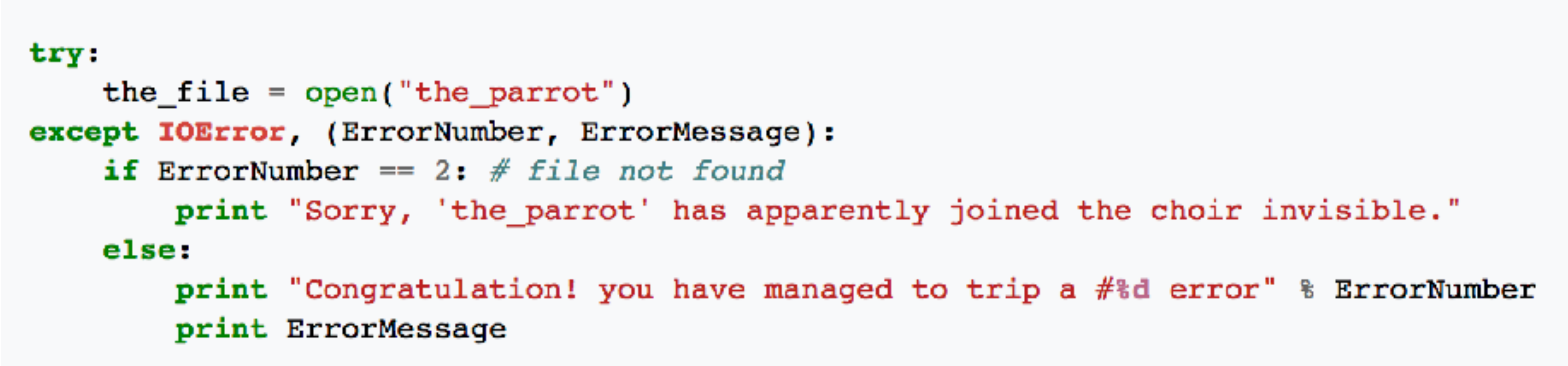
```
In [1]: try:
   ...:     print 1/0
   ...: except ZeroDivisionError:
   ...:     print "You can't divide by zero, you're silly."
   ...:
You can't divide by zero, you're silly.
```



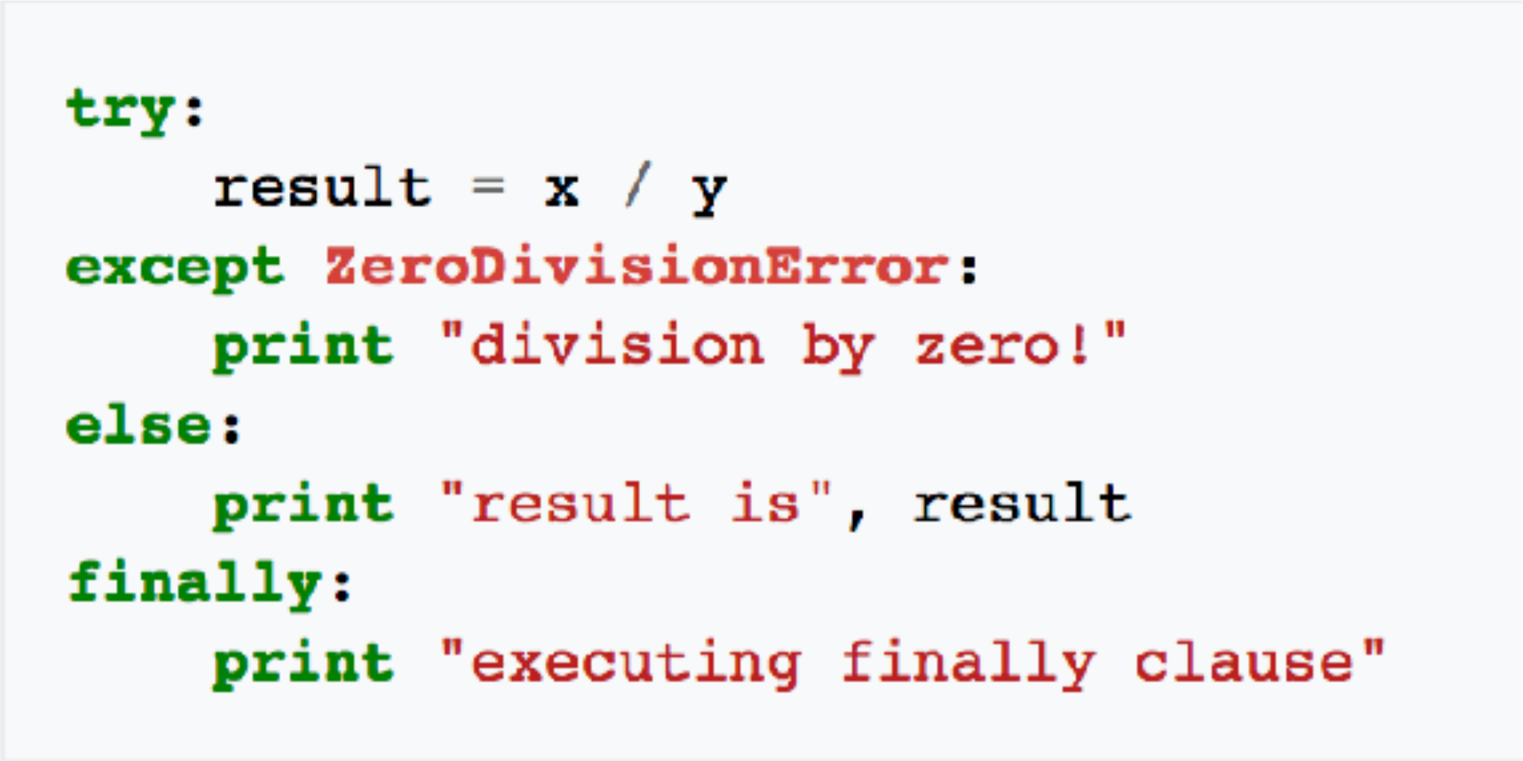
```
def f(x):
    return g(x) + 1

def g(x):
    if x < 0: raise ValueError, "I can't cope with a negative number here."
    else: return 5

try:
    print f(-6)
except ValueError:
    print "That value was invalid."
```

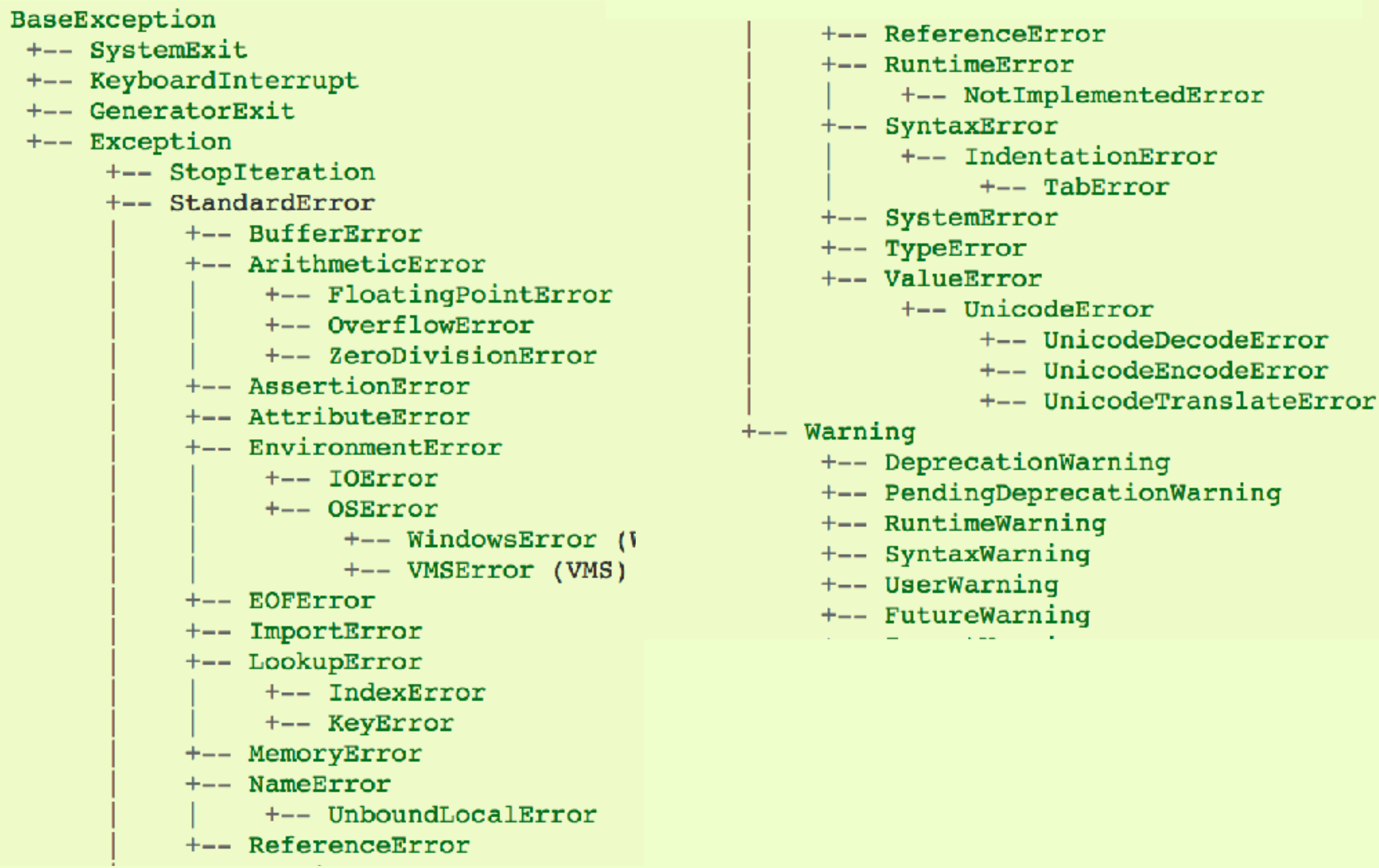


```
try:
    the_file = open("the_parrot")
except IOError, (ErrorNumber, ErrorMessage):
    if ErrorNumber == 2: # file not found
        print "Sorry, 'the_parrot' has apparently joined the choir invisible."
    else:
        print "Congratulation! you have managed to trip a #%d error" % ErrorNumber
        print ErrorMessage
```



```
try:
    result = x / y
except ZeroDivisionError:
    print "division by zero!"
else:
    print "result is", result
finally:
    print "executing finally clause"
```

Exception hierarchy



```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (W
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
```

```
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
```

```
#!/usr/bin/env python
#
#
# THIS CODE DOES NOT WORK

import sys

def read(a):
    myfile = open(a,'r'):
    for i in myfile:
        yield i
    myfile.close()

def count_chars(a):
    sum = 0
    for i in read(a):
        for j in i.split():
            sum += len(j)
    print sum

def count_chars2(a):
    i=''
    while (i != StopExecution):
        for j in i.split():
            sum += len(j)
    print sum


if __name__ == '__main__':
    total = count_chars(sys.argv[1]) + count_chars2(sys.argv[2])
    print total
```

```
nagal:L14>python simple.py frankenstein.txt alice.txt
  File "simple.py", line 9
    myfile = open(a,'r'):
                        ^
SyntaxError: invalid syntax
```

```
#!/usr/bin/env python
#
#
# THIS CODE DOES NOT WORK

import sys

def read(a):
    myfile = open(a,'r')
    for i in myfile:
        yield i
    myfile.close()

def count_chars(a):
    sum = 0
    for i in read(a):
        for j in i.split():
            sum += len(j)
    print sum

def count_chars2(a):
    i=''
    while (i != StopExecution):
        for j in i.split():
            sum += len(j)
    print sum


if __name__ == '__main__':
    total = count_chars(sys.argv[1]) + count_chars2(sys.argv[2])
    print total
```

```
nagal:L14>python simple1.py frankenstein.txt alice.txt
359240
Traceback (most recent call last):
  File "simple1.py", line 30, in <module>
    total = count_chars(sys.argv[1]) + count_chars2(sys.argv[2])
  File "simple1.py", line 23, in count_chars2
    while (i != StopExecution):
NameError: global name 'StopExecution' is not defined
```

```
#!/usr/bin/env python
#
#
# THIS CODE DOES NOT WORK

import sys

def read(a):
    myfile = open(a,'r')
    for i in myfile:
        yield i
    myfile.close()

def count_chars(a):
    sum = 0
    for i in read(a):
        for j in i.split():
            sum += len(j)
    print sum

def count_chars2(a):
    i=''
    while (i != StopExecution):
        for j in i.split():
            sum += len(j)
    print sum


if __name__ == '__main__':
    total = count_chars(sys.argv[1]) + count_chars2(sys.argv[2])
    print total
```

```
def count_chars2(a):
    return sum([len(j) for i in read(a) for j in 
i.split()])
```

```
WRONG:
def count_chars2(a):
    return sum([len(j) for j in i.split() for i in read(a)])
```

```
nagal:L14>python simple2.py frankenstein.txt alice.txt
Traceback (most recent call last):
  File "simple2.py", line 26, in <module>
    total = count_chars(sys.argv[1]) + count_chars2(sys.argv[2])
  File "simple2.py", line 22, in count_chars2
    return sum([j for j in i.split() for i in read(a)])
UnboundLocalError: local variable 'i' referenced before assignment
```

```
python –m pdb simple1.py frankenstein.txt alice.txt
```

Immutable Types Can't Be Changed in Place

Remember that you can't change an immutable object (e.g., tuple, string) in place:

```
T = (1, 2, 3)
T[2] = 4          # Error
```

Construct a new object with slicing, concatenation, and so on, and assign it back to the original variable if needed. Because Python automatically reclaims unused memory, this is not as wasteful as it may seem:

```
T = T[:2] + (4,)  # Okay: T becomes (1, 2, 4)
```

Use Simple `for` Loops Instead of `while` or `range`

When you need to step over all items in a sequence object from left to right, a simple `for` loop (e.g., `for x in seq:`) is simpler to code, and usually quicker to run, than a `while`- or `range`-based counter loop. Avoid the temptation to use `range` in a `for` unless you really have to; let Python handle the indexing for you. All three of the following loops work, but the first is usually better; in Python, simple is good.

```
S = "lumberjack"

for c in S:
    print c                   # simplest

for i in range(len(S)):
    print S[i]    # too much

i = 0                                 # too much
while i < len(S):
    print S[i]
    i += 1
```

Don't Expect Results From Functions That Change Objects

In-place change operations such as the `list.append( )` and `list.sort( )` methods modify an object, but do not return the object that was modified (they return `None`); call them without assigning the result. It's not uncommon for beginners to say something like:

```
mylist = mylist.append(X)
```

to try to get the result of an `append`; instead, this assigns `mylist` to `None`, rather than the modified list. A more devious example of this pops up when trying to step through dictionary items in sorted-key fashion:

```
D = {...}
for k in D.keys().sort():
    print D[k]
```

This almost works -- the `keys` method builds a `keys` list, and the `sort` method orders it -- but since the `sort` method returns `None`, the loop fails because it is ultimately a loop over `None` (a nonsequence). To code this correctly, split the method calls out into statements:

```
Ks = D.keys()
Ks.sort()
for k in Ks:
    print D[k]
```

Conversions Only Happen Among Number Types

In Python, an expression like `123 + 3.145` works -- it automatically converts the integer to a floating point, and uses floating point math. On the other hand, the following fails:

```
S = "42"
I = 1
X = S + I        # A type error
```

This is also on purpose, because it is ambiguous: should the string be converted to a number (for addition), or the number to a string (for concatenation)?. In Python, we say that explicit is better than implicit, so you must convert manually:

```
X = int(S) + I   # Do addition: 43
X = S + str(I)   # Do concatenation: "421"
```

Cyclic Datastructures Can Cause Loops

Although fairly rare in practice, if a collection object contains a reference to itself, it's called a *cyclic object*. Python prints a `[...]` whenever it detects a cycle in the object, rather than getting stuck in an infinite loop:

```
>>> L = ['grail']  # Append reference back to L
>>> L.append(L)    # Generates cycle in object
>>> L
['grail', [...]]
```

Besides understanding that the three dots represent a cycle in the object, this case is worth knowing about because cyclic structures may cause code of your own to fall into unexpected loops if you don't anticipate them. If needed, keep a list or dictionary of items already visited, and check it to know if you have reached a cycle.

Local Names Are Detected Statically

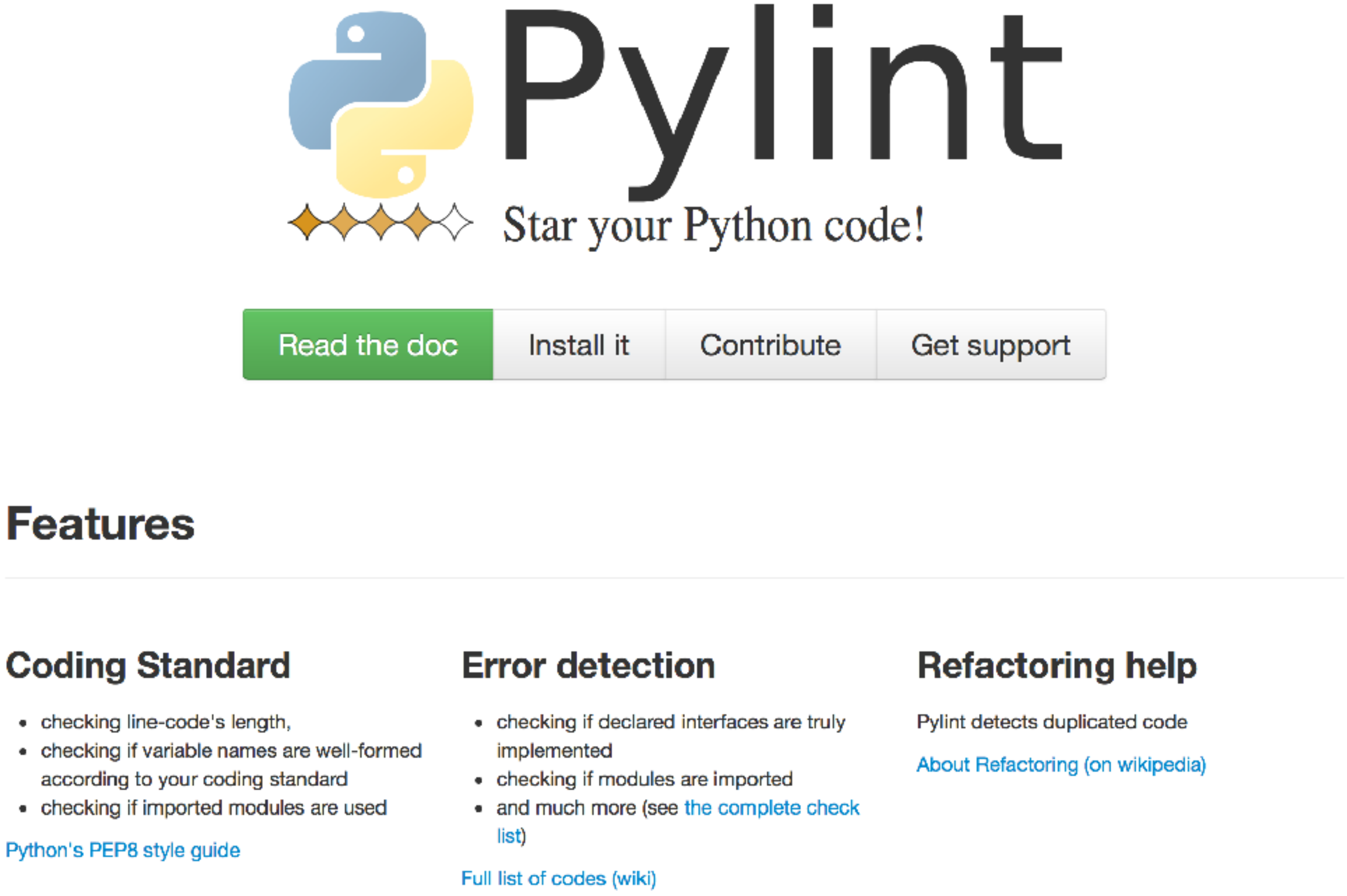
Python classifies names assigned in a function as locals by default; they live in the function's scope and exist only while the function is running. Technically, Python detects locals statically, when it compiles the `def`s code, rather than by noticing assignments as they happen at runtime. This can also lead to confusion if it's not understood. For example, watch what happens if you add an assignment to a variable after a reference:

```
>>> X = 99
>>> def func():
...     print X       # Does not yet exist
...     X = 88        # Makes X local in entire def
...
>>> func( )           # Error!
```

You get an undefined name error, but the reason is subtle. While compiling this code, Python sees the assignment to `X` and decides that `X` will be a local name everywhere in the function. But later, when the function is actually run, the assignment hasn't yet happened when the print executes, so Python raises an undefined name error.

Really, the previous example is ambiguous: did you mean to print the global `X` and then create a local `X`, or is this a genuine programming error? If you really mean to print global `X`, you need to declare it in a global statement, or reference it through the enclosing module name.

http://www.pylint.org



Features

Coding Standard

- checking line-code's length,
- checking if variable names are well-formed according to your coding standard
- checking if imported modules are used

Python's PEP8 style guide

Error detection

- checking if declared interfaces are truly implemented
- checking if modules are imported
- and much more (see the complete check list)

Full list of codes (wiki)

Refactoring help

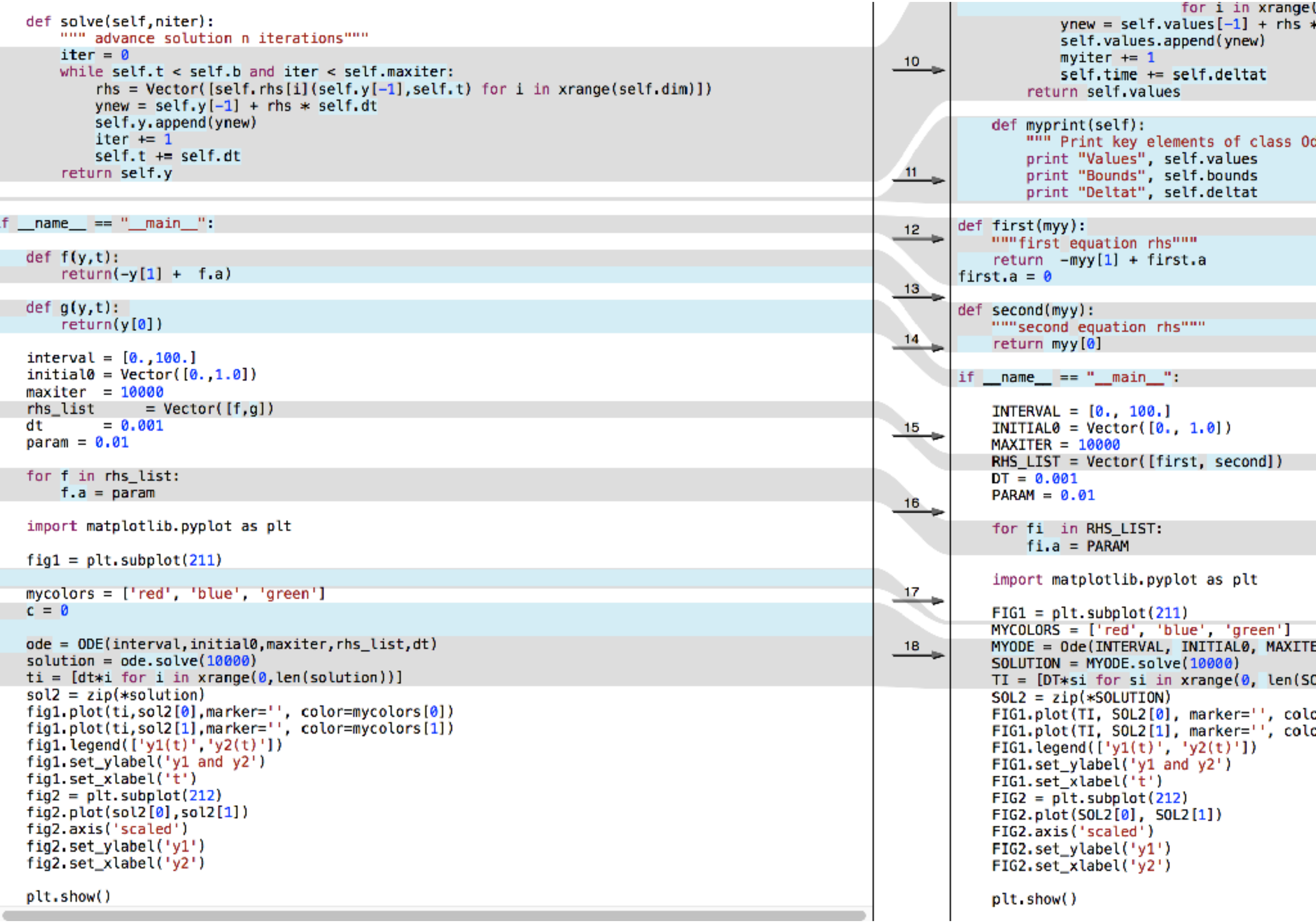
Pylint detects duplicated code

About Refactoring (on wikipedia)

```
nagal:L14>pylint node1.py
No config file found, using default configuration
************* Module node1
C:  9, 0: Exactly one space required after comma
    def __add__(self,other):
                    ^ (bad-whitespace)
C: 14, 0: Trailing whitespace (trailing-whitespace)
C: 19, 0: Exactly one space required after comma
    def __mul__(self,other):
                    ^ (bad-whitespace)
C: 27, 0: Exactly one space required after comma
    def __init__(self, interval,initial0,maxiter,rhs_list,dt):
                               ^ (bad-whitespace)
C: 27, 0: Exactly one space required after comma
    def __init__(self, interval,initial0,maxiter,rhs_list,dt):
                                        ^ (bad-whitespace)
C: 27, 0: Exactly one space required after comma
    def __init__(self, interval,initial0,maxiter,rhs_list,dt):
                                                ^ (bad-whitespace)
C: 27, 0: Exactly one space required after comma
    def __init__(self, interval,initial0,maxiter,rhs_list,dt):
                                                         ^ (bad-whitespace)
C: 29, 0: Exactly one space required after comma
        self.a,self.b = interval
              ^ (bad-whitespace)
C: 36, 0: Unnecessary parens after 'if' keyword (superfluous-parens)
C: 37, 0: Line too long (102/80) (line-too-long)
C: 37, 0: Exactly one space required after comma
            print "right hand side and initial values have inconsistent
length",self.dim,len(self.rhs)
                                                                               ^
(bad-whitespace)
```

```
C: 26, 0: Missing class docstring (missing-docstring)
W: 27,49: Redefining name 'rhs_list' from outer scope (line 63) (redefined-outer-name)
W: 27,23: Redefining name 'interval' from outer scope (line 60) (redefined-outer-name)
W: 27,41: Redefining name 'maxiter' from outer scope (line 62) (redefined-outer-name)
W: 27,58: Redefining name 'dt' from outer scope (line 64) (redefined-outer-name)
W: 27,32: Redefining name 'initial0' from outer scope (line 61) (redefined-outer-name)
R: 27, 4: Too many arguments (6/5) (too-many-arguments)
W: 44,61: Redefining name 'i' from outer scope (line 79) (redefined-outer-name)
W: 42, 8: Redefining built-in 'iter' (redefined-builtin)
W: 40,19: Unused argument 'niter' (unused-argument)
R: 26, 0: Too few public methods (1/2) (too-few-public-methods)
C: 54, 4: Invalid function name "f" (invalid-name)
C: 54, 4: Invalid argument name "y" (invalid-name)
C: 54, 4: Invalid argument name "t" (invalid-name)
C: 54, 4: Missing function docstring (missing-docstring)
E: 55,24: Function 'f' has no 'a' member (but some types could not be inferred) (maybe-no-
member)
W: 55,24: Using possibly undefined loop variable 'f' (undefined-loop-variable)
W: 54,12: Unused argument 't' (unused-argument)
```

```
            Global evaluation
            -----------------
            Your code has been rated at -4.46/10
```



```
   def solve(self,niter):
       """ advance solution n iterations"""
       iter = 0
       while self.t < self.b and iter < self.maxiter:
           rhs = Vector([self.rhs[i](self.y[-1],self.t) for i in xrange(self.dim)])
           ynew = self.y[-1] + rhs * self.dt
           self.y.append(ynew)
           iter += 1
           self.t += self.dt
       return self.y


f __name__ == "__main__":

   def f(y,t):
       return(-y[1] +  f.a)

   def g(y,t):
       return(y[0])

   interval = [0.,100.]
   initial0 = Vector([0.,1.0])
   maxiter  = 10000
   rhs_list      = Vector([f,g])
   dt       = 0.001
   param = 0.01

   for f in rhs_list:
       f.a = param

   import matplotlib.pyplot as plt

   fig1 = plt.subplot(211)

   mycolors = ['red', 'blue', 'green']
   c = 0

   ode = ODE(interval,initial0,maxiter,rhs_list,dt)
   solution = ode.solve(10000)
   ti = [dt*i for i in xrange(0,len(solution))]
   sol2 = zip(*solution)
   fig1.plot(ti,sol2[0],marker='', color=mycolors[0])
   fig1.plot(ti,sol2[1],marker='', color=mycolors[1])
   fig1.legend(['y1(t)','y2(t)'])
   fig1.set_ylabel('y1 and y2')
   fig1.set_xlabel('t')
   fig2 = plt.subplot(212)
   fig2.plot(sol2[0],sol2[1])
   fig2.axis('scaled')
   fig2.set_ylabel('y1')
   fig2.set_xlabel('y2')

   plt.show()
```

```
                          for i in xrange(
            ynew = self.values[-1] + rhs *
            self.values.append(ynew)
            myiter += 1
            self.time += self.deltat
        return self.values

    def myprint(self):
        """ Print key elements of class Od
        print "Values", self.values
        print "Bounds", self.bounds
        print "Deltat", self.deltat

def first(myy):
    """first equation rhs"""
    return  -myy[1] + first.a
first.a = 0

def second(myy):
    """second equation rhs"""
    return myy[0]

if __name__ == "__main__":

    INTERVAL = [0., 100.]
    INITIAL0 = Vector([0., 1.0])
    MAXITER = 10000
    RHS_LIST = Vector([first, second])
    DT = 0.001
    PARAM = 0.01

    for fi  in RHS_LIST:
        fi.a = PARAM

    import matplotlib.pyplot as plt

    FIG1 = plt.subplot(211)
    MYCOLORS = ['red', 'blue', 'green']
    MYODE = Ode(INTERVAL, INITIAL0, MAXITE
    SOLUTION = MYODE.solve(10000)
    TI = [DT*si for si in xrange(0, len(SO
    SOL2 = zip(*SOLUTION)
    FIG1.plot(TI, SOL2[0], marker='', colo
    FIG1.plot(TI, SOL2[1], marker='', colo
    FIG1.legend(['y1(t)', 'y2(t)'])
    FIG1.set_ylabel('y1 and y2')
    FIG1.set_xlabel('t')
    FIG2 = plt.subplot(212)
    FIG2.plot(SOL2[0], SOL2[1])
    FIG2.axis('scaled')
    FIG2.set_ylabel('y1')
    FIG2.set_xlabel('y2')

    plt.show()
```

```
nagal:L14>pylint node.py
No config file found, using default configuration
************* Module node
R: 32, 4: Too many arguments (6/5) (too-many-arguments)
W: 92,11: Used * or ** magic (star-args)


Report
======
72 statements analysed.

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

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |68     |75.56 |68       |=          |
+----------+-------+------+---------+-----------+
|docstring |14     |15.56 |14       |=          |
+----------+-------+------+---------+-----------+
|comment   |5      |5.56  |5        |=          |
+----------+-------+------+---------+-----------+
|empty     |3      |3.33  |3        |=          |
+----------+-------+------+---------+-----------+



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

+-----------+-------+---------+-----------+
|type       |number |previous |difference |
+===========+=======+=========+===========+
|convention |0      |0        |=          |
+-----------+-------+---------+-----------+
|refactor   |1      |1        |=          |
```