Fixing What 2to3 Can’t

WE'LL COVER THE FOLLOWING

False is invalid syntax

You do have tests, right?

Now for the real test: running the test harness against the test suite. Since the test suite is designed to cover all the possible code paths, it’s a good way to test our ported code to make sure there aren’t any bugs lurking anywhere.

```
C:\home\chardet> python test.py tests\*\*
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    from chardet.universaldetector import UniversalDetector
  File "C:\home\chardet\chardet\universaldetector.py", line 51
    self.done = constants.False
                              ^
SyntaxError: invalid syntax
```

Hmm, a small snag. In Python 3, `False` is a reserved word, so you can’t use it as a variable name. Let’s look at `constants.py` to see where it’s defined. Here’s as a variable name. Let's look at `constants.py` to see where it's defined. Here's the original version from `constants.py`, before the `2to3` script changed it:

```
import __builtin__
if not hasattr(__builtin__, 'False'):
    False = 0
    True = 1
else:
    False = __builtin__.False
    True = __builtin__.True
```

This piece of code is designed to allow this library to run under older versions of Python 2. Prior to Python 2.3, Python had no built-in `bool` type. This code detects the absence of the built-in constants `True` and `False`, and defines them if necessary.

However, Python 3 will always have a `bool` type, so this entire code snippet is unnecessary. The simplest solution is to replace all instances of `constants.True` and `constants.False` with `True` and `False`, respectively, then delete this dead code from `constants.py`.

So this line in `universaldetector.py`:

```
self.done = constants.False
```

Becomes

```
self.done = False
```

Ah, wasn't that satisfying? The code is shorter and more readable already.

No module named constants

Time to run `test.py` again and see how far it gets.

```
C:\home\chardet> python test.py tests\*\*
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    from chardet.universaldetector import UniversalDetector
  File "C:\home\chardet\chardet\universaldetector.py", line 29, in <module>
    import constants, sys
ImportError: No module named constants
```

What's that you say? No module named `constants`? Of course there's a module named constants. It’s right there, in `chardet/constants.py`.

Remember when the `2to3` script fixed up all those import statements? This library has a lot of relative imports — that is, modules that import other modules within the same library — but *the logic behind relative imports has changed in Python 3*. In Python 2, you could just `import constants` and it would look in the `chardet/` directory first. In Python 3, all import statements are absolute by default. If you want to do a relative import in Python 3, you need to be explicit about it:

```
from . import constants
```

But wait. Wasn’t the `2to3` script supposed to take care of these for you? Well, it did, but this particular import statement combines two different types of imports into one line: a relative import of the `constants` module within the library, and an absolute import of the `sys` module that is pre-installed in the Python standard library. In Python 2, you could combine these into one import statement. In Python 3, you can’t, and the `2to3` script is not smart enough to split the import statement into two.

The solution is to split the import statement manually. So this two-in-one import:

```
import constants, sys
```

Needs to become two separate imports:

```
from . import constants
import sys
```

There are variations of this problem scattered throughout the `chardet` library. In some places it’s “`import constants, sys`”; in other places, it’s “`import constants, re`”. The fix is the same: manually split the import statement into two lines, one for the relative import, the other for the absolute import.

Onward!

Name ‘file’ is not defined

open() is the new **file()**. PapayaWhip is the new black.

And here we go again, running `test.py` to try to execute our test cases…

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    for line in file(f, 'rb'):
NameError: name 'file' is not defined
```

This one surprised me, because I've been using this idiom as long as I can remember. In Python 2, the global `file()` function was an alias for the `open()` function, which was the standard way of opening text files for reading. In Python 3, the global `file()` function no longer exists, but the `open()` function still exists.

Thus, the simplest solution to the problem of the missing `file()` is to call the `open()` function instead:

```
for line in open(f, 'rb'):
```

And that's all I have to say about that.

Can't use a string pattern on a bytes-like object

Now things are starting to get interesting. And by "interesting," I mean "confusing as all hell."

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 98, in feed
    if self._highBitDetector.search(aBuf):
TypeError: can't use a string pattern on a bytes-like object
```

To debug this, let's see what `self._highBitDetector` is. It's defined in the `__init__` method of the `UniversalDetector` class:

```
class UniversalDetector:
    def __init__(self):
        self._highBitDetector = re.compile(r'[\x80-\xFF]')
```

This pre-compiles a regular expression designed to find non-ascii characters in the range 128–255 (0x80–0xFF). Wait, that's not quite right; I need to be more precise with my terminology. This pattern is designed to find non-ascii bytes in the range 128-255.

And therein lies the problem.

In Python 2, a string was an array of bytes whose character encoding was tracked separately. If you wanted Python 2 to keep track of the character encoding, you had to use a Unicode string (`u`'') instead. But in Python 3, a string is always what Python 2 called a Unicode string — that is, an array of Unicode characters (of possibly varying byte lengths). Since this regular expression is defined by a string pattern, it can only be used to search a string — again, an array of characters. But what we're searching is not a string, it's a byte array. Looking at the traceback, this error occurred in `universaldetector.py`:

```
def feed(self, aBuf):
    #.
    #.
    #.
    if self._mInputState == ePureAscii:
        if self._highBitDetector.search(aBuf):
          pass
```

And what is `aBuf`? Let's backtrack further to a place that calls `UniversalDetector.feed()`. One place that calls it is the test harness, `test.py`.

```
u = UniversalDetector()
#.
#.
#.
for line in open(f, 'rb'):
    u.feed(line)
```

And here we find our answer: in the `UniversalDetector.feed()` method, `aBuf` is a line read from a file on disk. Look carefully at the parameters used to open the file: `'rb'. 'r'` is for "read"; OK, big deal, we're reading the file. Ah, but 'b' is for "binary." Without the `'b'` flag, this `for` loop would read the file, line by line, and convert each line into a string — an array of Unicode characters — according to the system default character encoding. But with the `'b'` flag, this `for` loop reads the file, line by line, and stores each line exactly as it appears in the file, as an array of bytes. That byte array gets passed to `UniversalDetector.feed()`, and eventually gets passed to the pre-compiled regular expression, `self._highBitDetector`, to search for high-bit… characters. But we don’t have characters; we have bytes. Oops.

Not an array of characters, but an array of bytes.

What we need this regular expression to search is not an array of characters, but an array of bytes.

Once you realize that, the solution is not difficult. Regular expressions defined with strings can search strings. Regular expressions defined with byte arrays can search byte arrays. To define a byte array pattern, we simply change the type of the argument we use to define the regular expression to a byte array. (There is one other case of this same problem, on the very next line.)

```
class UniversalDetector:
      def __init__(self):
-         self._highBitDetector = re.compile(r'[\x80-\xFF]')
-         self._escDetector = re.compile(r'(\033|~{)')
+         self._highBitDetector = re.compile(b'[\x80-\xFF]')
+         self._escDetector = re.compile(b'(\033|~{)')
          self._mEscCharSetProber = None
          self._mCharSetProbers = []
          self.reset()
```

Searching the entire codebase for other uses of the re module turns up two more instances, in `charsetprober.py`. Again, the code is defining regular expressions as strings but executing them on `aBuf`, which is a byte array. The solution is the same: define the regular expression patterns as byte arrays.

```
class CharSetProber:
      .
      .
      .
      def filter_high_bit_only(self, aBuf):
-         aBuf = re.sub(r'([\x00-\x7F])+', ' ', aBuf)
+         aBuf = re.sub(b'([\x00-\x7F])+', b' ', aBuf)
          return aBuf

      def filter_without_english_letters(self, aBuf):
-         aBuf = re.sub(r'([A-Za-z])+', ' ', aBuf)
+         aBuf = re.sub(b'([A-Za-z])+', b' ', aBuf)
          return aBuf
```

Can’t convert ‘bytes’ object to str implicitly

Curiouser and curiouser…

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 100, in feed
    elif (self._mInputState == ePureAscii) and self._escDetector.search(self._mLastChar + aBu
TypeError: Can't convert 'bytes' object to str implicitly
```

There’s an unfortunate clash of coding style and Python interpreter here. The `TypeError` could be anywhere on that line, but the traceback doesn’t tell you exactly where it is. It could be in the first conditional or the second, and the traceback would look the same. To narrow it down, you should split the line in half, like this:

```
elif (self._mInputState == ePureAscii) and \
    self._escDetector.search(self._mLastChar + aBuf):
```

And re-run the test:

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 101, in feed
    self._escDetector.search(self._mLastChar + aBuf):
TypeError: Can't convert 'bytes' object to str implicitly
```

Aha! The problem was not in the first conditional (`self._mInputState == ePureAscii`) but in the second one. So what could cause a `TypeError` there? Perhaps you’re thinking that the `search()` method is expecting a value of a different type, but that wouldn’t generate this traceback. Python functions can take any value; if you pass the right number of arguments, the function will execute. It may *crash* if you pass it a value of a different type than it’s expecting, but if that happened, the traceback would point to somewhere inside the function. But this traceback says it never got as far as calling the `search()` method. So the problem must be in that `+` operation, as it’s trying to construct the value that it will eventually pass to the `search()` method.

We know from previous debugging that `aBuf` is a byte array. So what is `self._mLastChar?` It’s an instance variable, defined in the `reset()` method, which is actually called from the `__init__()` method.

```
class UniversalDetector:
    def __init__(self):
        self._highBitDetector = re.compile(b'[\x80-\xFF]')
        self._escDetector = re.compile(b'(\033|~{)')
        self._mEscCharSetProber = None
        self._mCharSetProbers = []
        self.reset()

    def reset(self):
        self.result = {'encoding': None, 'confidence': 0.0}
        self.done = False
        self._mStart = True
        self._mGotData = False
        self._mInputState = ePureAscii
        self._mLastChar = ''
```

And now we have our answer. Do you see it? `self._mLastChar` is a string, but `aBuf` is a byte array. And you can’t concatenate a string to a byte array — not even a zero-length string.

So what is `self._mLastChar` anyway? In the `feed()` method, just a few lines down from where the trackback occurred.

```
if self._mInputState == ePureAscii:
    if self._highBitDetector.search(aBuf):
        self._mInputState = eHighbyte
    elif (self._mInputState == ePureAscii) and \
            self._escDetector.search(self._mLastChar + aBuf):
        self._mInputState = eEscAscii

self._mLastChar = aBuf[-1]
```

The calling function calls this `feed()` method over and over again with a few bytes at a time. The method processes the bytes it was given (passed in as `aBuf`), then stores the last byte in `self._mLastChar` in case it’s needed during the next call. (In a multi-byte encoding, the `feed()` method might get called with half of a character, then called again with the other half.) But because `aBuf` is now a byte array instead of a string, `self._mLastChar` needs to be a byte array as well. Thus:

```
def reset(self):
      .

      .
      .
-     self._mLastChar = ''
+     self._mLastChar = b''
```

Searching the entire codebase for “`mLastChar`” turns up a similar problem in `mbcharsetprober.py`, but instead of tracking the last character, it tracks the last *two* characters. The `MultiByteCharSetProber` class uses a list of 1-character strings to track the last two characters. In Python 3, it needs to use a list of integers, because it’s not really tracking characters, it’s tracking bytes. (Bytes are just integers from 0-255.)

```
class MultiByteCharSetProber(CharSetProber):
      def __init__(self):
          CharSetProber.__init__(self)
          self._mDistributionAnalyzer = None
          self._mCodingSM = None
-         self._mLastChar = ['\x00', '\x00']
+         self._mLastChar = [0, 0]

      def reset(self):
          CharSetProber.reset(self)
          if self._mCodingSM:
              self._mCodingSM.reset()
          if self._mDistributionAnalyzer:
              self._mDistributionAnalyzer.reset()
-         self._mLastChar = ['\x00', '\x00']
+         self._mLastChar = [0, 0]
```

Unsupported operand type(s) for +: ‘int’ and ‘bytes’

I have good news, and I have bad news. The good news is we’re making progress…

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 101, in feed
    self._escDetector.search(self._mLastChar + aBuf):
TypeError: unsupported operand type(s) for +: 'int' and 'bytes'
```

…The bad news is it doesn’t always feel like progress.

But this is progress! Really! Even though the traceback calls out the same line of code, it’s a different error than it used to be. Progress! So what’s the problem now? The last time I checked, this line of code didn’t try to concatenate an `int` with a byte array (`bytes`). In fact, you just spent a lot of time ensuring that `self._mLastChar` was a byte array. How did it turn into an `int`?

The answer lies not in the previous lines of code, but in the following lines.

```
if self._mInputState == ePureAscii:
    if self._highBitDetector.search(aBuf):
        self._mInputState = eHighbyte
    elif (self._mInputState == ePureAscii) and \
            self._escDetector.search(self._mLastChar + aBuf):
        self._mInputState = eEscAscii

self._mLastChar = aBuf[-1]
```

Each item in a string is a string. Each item in a byte array is an integer.

This error doesn’t occur the first time the `feed()` method gets called; it occurs the second time, after `self._mLastChar` has been set to the last byte of `aBuf`. Well, what’s the problem with that? Getting a single element from a byte array yields an integer, not a byte array. To see the difference, follow me to the interactive shell:

```
aBuf = b'\xEF\xBB\xBF'                 #①
print (len(aBuf))
#3

mLastChar = aBuf[-1]
print (mLastChar )                     #②
#191

print (type(mLastChar) )               #③
#<class 'int'>

print (mLastChar + aBuf)               #④
#Traceback (most recent call last):
#   File "/usercode/__ed_file.py", line 12, in <module>
# print (mLastChar + aBuf) #\u2463
#TypeError: unsupported operand type(s) for +: 'int' and 'bytes'
```

```
aBuf = b'\xEF\xBB\xBF'

mLastChar = aBuf[-1:]          #⑤
print (mLastChar)
#b'\xbf'

print (mLastChar + aBuf)       #⑥
#b'\xbf\xef\xbb\xbf'
```

① Define a byte array of length 3.

② The last element of the byte array is 191.

③ That's an integer.

④ Concatenating an integer with a byte array doesn't work. You've now replicated the error you just found in `universaldetector.py`.

⑤ Ah, here's the fix. Instead of taking the last element of the byte array, use list slicing to create a new byte array containing just the last element. That is, start with the last element and continue the slice until the end of the byte array. Now `mLastChar` is a byte array of length 1.

⑥ Concatenating a byte array of length 1 with a byte array of length 3 returns a new byte array of length 4.

So, to ensure that the `feed()` method in `universaldetector.py` continues to work no matter how often it's called, you need to initialize `self._mLastChar` as a 0-length byte array, then *make sure it stays a byte array*.

```
   self._escDetector.search(self._mLastChar + aBuf):
          self._mInputState = eEscAscii

- self._mLastChar = aBuf[-1]
+ self._mLastChar = aBuf[-1:]
```

ord() expected string of length 1, but int found

Tired yet? You're almost there…

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml                       ascii with confidence 1.0

tests\Big5\0804.blogspot.com.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 116, in feed
    if prober.feed(aBuf) == constants.eFoundIt:
  File "C:\home\chardet\chardet\charsetgroupprober.py", line 60, in feed
    st = prober.feed(aBuf)
  File "C:\home\chardet\chardet\utf8prober.py", line 53, in feed
    codingState = self._mCodingSM.next_state(c)
  File "C:\home\chardet\chardet\codingstatemachine.py", line 43, in next_state
    byteCls = self._mModel['classTable'][ord(c)]
TypeError: ord() expected string of length 1, but int found
```

OK, so `c` is an `int`, but the `ord()` function was expecting a 1-character string. Fair enough. Where is `c` defined?

```
# codingstatemachine.py
def next_state(self, c):
    # for each byte we get its class
    # if it is first byte, we also get byte length
    byteCls = self._mModel['classTable'][ord(c)]
```

That's no help; it's just passed into the function. Let's pop the stack.

```
# utf8prober.py
def feed(self, aBuf):
    for c in aBuf:
        codingState = self._mCodingSM.next_state(c)
```

Do you see it? In Python 2, `aBuf` was a string, so `c` was a 1-character string. (That's what you get when you iterate over a string — all the characters, one by one.) But now, `aBuf` is a byte array, so `c` is an int, not a 1-character string. In other words, there's no need to call the `ord()` function because `c` is already an `int`!

Thus:

```
def next_state(self, c):
      # for each byte we get its class
      # if it is first byte, we also get byte length
-     byteCls = self._mModel['classTable'][ord(c)]
+     byteCls = self._mModel['classTable'][c]
```

Searching the entire codebase for instances of "`ord(c)`" uncovers similar

Searching the entire codebase for instances of `ord(c)` uncovers similar problems in `sbcharsetprober.py`…

```
# sbcharsetprober.py
def feed(self, aBuf):
    if not self._mModel['keepEnglishLetter']:
        aBuf = self.filter_without_english_letters(aBuf)
    aLen = len(aBuf)
    if not aLen:
        return self.get_state()
    for c in aBuf:
        order = self._mModel['charToOrderMap'][ord(c)]
```

…and `latin1prober.py`…

```
# latin1prober.py
def feed(self, aBuf):
    aBuf = self.filter_with_english_letters(aBuf)
    for c in aBuf:
        charClass = Latin1_CharToClass[ord(c)]
```

`c` is iterating over `aBuf`, which means it is an integer, not a 1-character string. The solution is the same: change `ord(c)` to just plain `c`.

```
 # sbcharsetprober.py
  def feed(self, aBuf):
      if not self._mModel['keepEnglishLetter']:
          aBuf = self.filter_without_english_letters(aBuf)
      aLen = len(aBuf)
      if not aLen:
          return self.get_state()
      for c in aBuf:
-         order = self._mModel['charToOrderMap'][ord(c)]
+         order = self._mModel['charToOrderMap'][c]

  # latin1prober.py
  def feed(self, aBuf):
      aBuf = self.filter_with_english_letters(aBuf)
      for c in aBuf:
-         charClass = Latin1_CharToClass[ord(c)]
+         charClass = Latin1_CharToClass[c]
```

Unorderable types: int() >= str()

Let's go again.

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml                       ascii with confidence 1.0
tests\Big5\0804.blogspot.com.xml
Traceback (most recent call last):
  File "test.py", line 10, in <module>
```

```
    u.feed(line)
  File "C:\home\chardet\chardet\universaldetector.py", line 116, in feed
    if prober.feed(aBuf) == constants.eFoundIt:
  File "C:\home\chardet\chardet\charsetgroupprober.py", line 60, in feed
    st = prober.feed(aBuf)
  File "C:\home\chardet\chardet\sjisprober.py", line 68, in feed
    self._mContextAnalyzer.feed(self._mLastChar[2 - charLen :], charLen)
  File "C:\home\chardet\chardet\jpcntx.py", line 145, in feed
    order, charLen = self.get_order(aBuf[i:i+2])
  File "C:\home\chardet\chardet\jpcntx.py", line 176, in get_order
    if ((aStr[0] >= '\x81') and (aStr[0] <= '\x9F')) or \
TypeError: unorderable types: int() >= str()
```

So what's this all about? "Unorderable types"? Once again, the difference between byte arrays and strings is rearing its ugly head. Take a look at the code:

```
class SJISContextAnalysis(JapaneseContextAnalysis):
    def get_order(self, aStr):
        if not aStr: return -1, 1
        # find out current char's byte length
        if ((aStr[0] >= '\x81') and (aStr[0] <= '\x9F')) or \
           ((aStr[0] >= '\xE0') and (aStr[0] <= '\xFC')):
            charLen = 2
        else:
            charLen = 1
```

And where does `aStr` come from? Let's pop the stack:

```
def feed(self, aBuf, aLen):
    #.
    #.
    #.
    i = self._mNeedToSkipCharNum
    while i < aLen:
        order, charLen = self.get_order(aBuf[i:i+2])
```

Oh look, it's our old friend, `aBuf`. As you might have guessed from every other issue we've encountered in this chapter, `aBuf` is a byte array. Here, the `feed()` method isn't just passing it on wholesale; it's slicing it. But as you saw earlier in this chapter, slicing a byte array returns a byte array, so the `aStr` parameter that gets passed to the `get_order()` method is still a byte array.

And what is this code trying to do with `aStr`? It's taking the first element of the byte array and comparing it to a string of length 1. In Python 2, that worked, because `aStr` and `aBuf` were strings, and `aStr[0]` would be a string, and you can compare strings for inequality. But in Python 3, `aStr` and `aBuf` are byte arrays, `aStr[0]` is an integer, and you can't compare integers and strings for inequality without explicitly coercing one of them.

In this case, there's no need to make the code more complicated by adding an explicit coercion. `aStr[0]` yields an integer; the things you're comparing to are all constants. Let's change them from 1-character strings to integers. And while we're at it, let's change `aStr` to `aBuf`, since it's not actually a string.

```
class SJISContextAnalysis(JapaneseContextAnalysis):
-     def get_order(self, aStr):
-      if not aStr: return -1, 1
+     def get_order(self, aBuf):
+      if not aBuf: return -1, 1
         # find out current char's byte length
-        if ((aStr[0] >= '\x81') and (aStr[0] <= '\x9F')) or \
-           ((aBuf[0] >= '\xE0') and (aBuf[0] <= '\xFC')):
+        if ((aBuf[0] >= 0x81) and (aBuf[0] <= 0x9F)) or \
+           ((aBuf[0] >= 0xE0) and (aBuf[0] <= 0xFC)):
             charLen = 2
         else:
             charLen = 1

         # return its order if it is hiragana
-      if len(aStr) > 1:
-             if (aStr[0] == '\202') and \
-                (aStr[1] >= '\x9F') and \
-                (aStr[1] <= '\xF1'):
-                return ord(aStr[1]) - 0x9F, charLen
+      if len(aBuf) > 1:
+             if (aBuf[0] == 202) and \
+                (aBuf[1] >= 0x9F) and \
+                (aBuf[1] <= 0xF1):
+                return aBuf[1] - 0x9F, charLen

         return -1, charLen

  class EUCJPContextAnalysis(JapaneseContextAnalysis):
-     def get_order(self, aStr):
-      if not aStr: return -1, 1
+     def get_order(self, aBuf):
+      if not aBuf: return -1, 1
         # find out current char's byte length
-        if (aStr[0] == '\x8E') or \
-          ((aStr[0] >= '\xA1') and (aStr[0] <= '\xFE')):
+        if (aBuf[0] == 0x8E) or \
+          ((aBuf[0] >= 0xA1) and (aBuf[0] <= 0xFE)):
             charLen = 2
-        elif aStr[0] == '\x8F':
+        elif aBuf[0] == 0x8F:
             charLen = 3
         else:
             charLen = 1

       # return its order if it is hiragana
-     if len(aStr) > 1:
-           if (aStr[0] == '\xA4') and \
-              (aStr[1] >= '\xA1') and \
-              (aStr[1] <= '\xF3'):
```

```
-                  return ord(aStr[1]) - 0xA1, charLen
+    if len(aBuf) > 1:
+           if (aBuf[0] == 0xA4) and \

+              (aBuf[1] >= 0xA1) and \
+              (aBuf[1] <= 0xF3):
+               return aBuf[1] - 0xA1, charLen

         return -1, charLen
```

Searching the entire codebase for occurrences of the `ord()` function uncovers the same problem in `chardistribution.py` (specifically, in the `EUCTWDistributionAnalysis`, `EUCKRDistributionAnalysis`, `GB2312DistributionAnalysis`, `Big5DistributionAnalysis`, `SJISDistributionAnalysis`, and `EUCJPDistributionAnalysis` classes. In each case, the fix is similar to the change we made to the `EUCJPContextAnalysis` and `SJISContextAnalysis` classes in `jpcntx.py`.

Global name 'reduce' is not defined

Once more into the breach…

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml                       ascii with confidence 1.0
tests\Big5\0804.blogspot.com.xml
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    u.close()
  File "C:\home\chardet\chardet\universaldetector.py", line 141, in close
    proberConfidence = prober.get_confidence()
  File "C:\home\chardet\chardet\latin1prober.py", line 126, in get_confidence
    total = reduce(operator.add, self._mFreqCounter)
NameError: global name 'reduce' is not defined
```

According to the official What's New In Python 3.0 guide, the `reduce()` function has been moved out of the global namespace and into the `functools` module. Quoting the guide: "Use `functools.reduce()` if you really need it; however, 99 percent of the time an explicit `for` loop is more readable." You can read more about the decision from Guido van Rossum's weblog: The fate of reduce() in Python 3000.

```
def get_confidence(self):
    if self.get_state() == constants.eNotMe:
        return 0.01

    total = reduce(operator.add, self._mFreqCounter)
```

The `reduce()` function takes two arguments — a function and a list (strictly speaking, any iterable object will do) — and applies the function cumulatively to each item of the list. In other words, this is a fancy and roundabout way of adding up all the items in a list and returning the result.

This monstrosity was so common that Python added a global `sum()` function.

```
 def get_confidence(self):
      if self.get_state() == constants.eNotMe:
          return 0.01

-     total = reduce(operator.add, self._mFreqCounter)
+     total = sum(self._mFreqCounter)
```

Since you're no longer using the `operator` module, you can remove that `import` from the top of the file as well.

```
 from .charsetprober import CharSetProber
  from . import constants
- import operator
```

I CAN HAZ TESTZ?

```
C:\home\chardet> python test.py tests\*\*
tests\ascii\howto.diveintomark.org.xml                       ascii with confidence 1.0
tests\Big5\0804.blogspot.com.xml                             Big5 with confidence 0.99
tests\Big5\blog.worren.net.xml                               Big5 with confidence 0.99
tests\Big5\carbonxiv.blogspot.com.xml                        Big5 with confidence 0.99
tests\Big5\catshadow.blogspot.com.xml                        Big5 with confidence 0.99
tests\Big5\coolloud.org.tw.xml                               Big5 with confidence 0.99
tests\Big5\digitalwall.com.xml                               Big5 with confidence 0.99
tests\Big5\ebao.us.xml                                       Big5 with confidence 0.99
tests\Big5\fudesign.blogspot.com.xml                         Big5 with confidence 0.99
tests\Big5\kafkatseng.blogspot.com.xml                       Big5 with confidence 0.99
tests\Big5\ke207.blogspot.com.xml                            Big5 with confidence 0.99
tests\Big5\leavesth.blogspot.com.xml                         Big5 with confidence 0.99
tests\Big5\letterlego.blogspot.com.xml                       Big5 with confidence 0.99
tests\Big5\linyijen.blogspot.com.xml                         Big5 with confidence 0.99
tests\Big5\marilynwu.blogspot.com.xml                        Big5 with confidence 0.99
tests\Big5\myblog.pchome.com.tw.xml                          Big5 with confidence 0.99
tests\Big5\oui-design.com.xml                                Big5 with confidence 0.99
tests\Big5\sanwenji.blogspot.com.xml                         Big5 with confidence 0.99
tests\Big5\sinica.edu.tw.xml                                 Big5 with confidence 0.99
tests\Big5\sylvia1976.blogspot.com.xml                       Big5 with confidence 0.99
tests\Big5\tlkkuo.blogspot.com.xml                           Big5 with confidence 0.99
tests\Big5\tw.blog.xubg.com.xml                              Big5 with confidence 0.99
tests\Big5\unoriginalblog.com.xml                            Big5 with confidence 0.99
tests\Big5\upsaid.com.xml                                    Big5 with confidence 0.99
tests\Big5\willythecop.blogspot.com.xml                      Big5 with confidence 0.99
tests\Big5\ytc.blogspot.com.xml                              Big5 with confidence 0.99
tests\EUC-JP\aivy.co.jp.xml                                  EUC-JP with confidence 0.99
```

```
tests\EUC-JP\akaname.main.jp.xml                             EUC-JP with confidence 0.99
tests\EUC-JP\arclamp.jp.xml                                  EUC-JP with confidence 0.99
.

.
.
316 tests
```

Holy crap, it actually works! /me does a little dance