```python
"""A test script to compare MD5 implementations.

A note about performance: the pure Python MD5 takes roughly
160 sec. per MB of data on a 233 MHz Intel Pentium CPU.
"""

import string, unittest
import md5    # Python's implementation in C.
import md5py  # The pure Python implementation.


# Helpers...

def formatHex(str):
    "Print a string's HEX code in groups of two digits."

    d = map(None, str)
    d = map(ord, d)
    d = map(lambda x:"%02x" % x, d)
    return string.join(d, " ")


def format(str):
    "Print a string as-is in groups of two characters."

    s = ''
    for i in range(0, len(str)-1, 2):
        s = s + "%03s" % str[i:i+2]
    return s[1:]


def printDiff(message, d1, d2, expectedResult=None):
    "Print different outputs for same message."

    print "Message: '%s'" % message
    print "Message length: %d" % len(message)
    if expectedResult:
        print "%-48s (expected)" % format(expectedResult)
    print "%-48s (Std. lib. MD5)" % formatHex(d1)
    print "%-48s (Pure Python MD5)" % formatHex(d2)
    print


# The real comparison function.

def compareImp(message):
    """Compare two MD5 implementations, C vs. pure Python module.

    For equal digests this returns None, otherwise it returns
    a tuple of both digests.
    """

    # Use Python's standard library MD5 compiled C module.
    m1 = md5.md5()
    m1.update(message)
    d1 = m1.digest()
    d1h = m1.hexdigest()

    # Use MD5 module in pure Python.
    m2 = md5py.md5()
    m2.update(message)
    d2 = m2.digest()
    d2h = m2.hexdigest()

    # Return None if equal or the different digests if not equal.
    if d1 == d2 and d1h == d2h:
        return
    else:
        return d1, d2
```

```
class MD5CompareTestCase(unittest.TestCase):
    "Compare pure Python MD5 against Python's std. lib. version."

    def test1(self):
        "Test cases with known digest result."

        cases = (
          ("",
           "d41d8cd98f00b204e9800998ecf8427e"),
          ("a",
           "0cc175b9c0f1b6a831c399e269772661"),
          ("abc",
           "900150983cd24fb0d6963f7d28e17f72"),
          ("message digest",
           "f96b697d7cb7938d525a2f31aaf161d0"),
          ("abcdefghijklmnopqrstuvwxyz",
           "c3fcd3d76192e4007dfb496cca67e13b"),
          ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
           "d174ab98d277d9f5a5611c2c9f419d9f"),
          ("1234567890"*8,
           "57edf4a22be3c955ac49da2e2107b67a"),
        )

        for i in xrange(len(cases)):
            res = compareImp(cases[i][0])
            try:
                assert res == None
            except AssertionError, details:
                d1, d2 = res
                message, expectedResult = cases[i][0], None
                if len(cases[i]) == 2:
                    expectedResult = cases[i][1]
                printDiff(message, d1, d2, expectedResult)


    def test2(self):
        "Test cases without known digest result."

        cases = (
          "123",
          "1234",
          "12345",
          "123456",
          "1234567",
          "12345678",
          "123456789 123456789 123456789 ",
          "123456789 123456789 ",
          "123456789 123456789 1",
          "123456789 123456789 12",
          "123456789 123456789 123",
          "123456789 123456789 1234",
          "123456789 123456789 123456789 1",
          "123456789 123456789 123456789 12",
          "123456789 123456789 123456789 123",
          "123456789 123456789 123456789 1234",
          "123456789 123456789 123456789 12345",
          "123456789 123456789 123456789 123456",
          "123456789 123456789 123456789 1234567",
          "123456789 123456789 123456789 12345678",
         )

        for i in xrange(len(cases)):
            res = compareImp(cases[i][0])
            try:
                assert res == None
            except AssertionError, details:
                d1, d2 = res
                message = cases[i][0]
                printDiff(message, d1, d2)
    def test3(self):
        "Test cases with long messages (can take a while)."

        cases = (
          (2**10*'a',),
          (2**10*'abcd',),
##          (2**20*'a',),  ## 1 MB, takes about 160 sec. on a 233 Mhz Pentium.
         )

        for i in xrange(len(cases)):
            res = compareImp(cases[i][0])
            try:
                assert res == None
            except AssertionError, details:
                d1, d2 = res
                message = cases[i][0]
                printDiff(message, d1, d2)


    def test4(self):
        "Test cases with increasingly growing message lengths."

        i = 0
        while i  < 2**5:
            message = i * 'a'
            res = compareImp(message)
            try:
                assert res == None
            except AssertionError, details:
                d1, d2 = res
                printDiff(message, d1, d2)
            i = i + 1


    def test5(self):
        "Test updating cloned objects."

        cases = (
          "123",
          "1234",
          "12345",
          "123456",
          "1234567",
          "12345678",
          "123456789 123456789 123456789 ",
          "123456789 123456789 ",
          "123456789 123456789 1",
          "123456789 123456789 12",
          "123456789 123456789 123",
          "123456789 123456789 1234",
          "123456789 123456789 123456789 1",
          "123456789 123456789 123456789 12",
          "123456789 123456789 123456789 123",
          "123456789 123456789 123456789 1234",
          "123456789 123456789 123456789 12345",
          "123456789 123456789 123456789 123456",
          "123456789 123456789 123456789 1234567",
          "123456789 123456789 123456789 12345678",
         )

        # Load both with same prefix.
        prefix1 = 2**10 * 'a'

        m1 = md5.md5()
        m1.update(prefix1)
        m1c = m1.copy()

        m2 = md5py.md5()
        m2.update(prefix1)
        m2c = m2.copy()

        # Update and compare...
        for i in xrange(len(cases)):
            message = cases[i][0]

            m1c.update(message)
            d1 = m1c.hexdigest()

            m2c.update(message)
            d2 = m2c.hexdigest()

            assert d1 == d2


def makeSuite():
    suite = unittest.TestSuite()

    suite.addTest(MD5CompareTestCase('test1'))
    suite.addTest(MD5CompareTestCase('test2'))
    suite.addTest(MD5CompareTestCase('test3'))
    suite.addTest(MD5CompareTestCase('test4'))
    suite.addTest(MD5CompareTestCase('test5'))

    return suite


if __name__ == "__main__":
    unittest.TextTestRunner().run(makeSuite())
```