difflib — Compare Sequences

Purpose: Compare sequences, especially lines of text.

The `difflib` module contains tools for computing and working with differences between sequences. It is especially useful for comparing text, and includes functions that produce reports using several common difference formats.

The examples in this section will all use this common test data in the `difflib_data.py` module:

```
# difflib_data.py

text1 = """Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Integer eu lacus accumsan arcu fermentum euismod. Donec
pulvinar porttitor tellus. Aliquam venenatis. Donec facilisis
pharetra tortor.  In nec mauris eget magna consequat
convalis. Nam sed sem vitae odio pellentesque interdum. Sed
consequat viverra nisl. Suspendisse arcu metus, blandit quis,
rhoncus ac, pharetra eget, velit. Mauris urna. Morbi nonummy
molestie orci. Praesent nisi elit, fringilla ac, suscipit non,
tristique vel, mauris. Curabitur vel lorem id nisl porta
adipiscing. Suspendisse eu lectus. In nunc. Duis vulputate
tristique enim. Donec quis lectus a justo imperdiet tempus."""

text1_lines = text1.splitlines()

text2 = """Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Integer eu lacus accumsan arcu fermentum euismod. Donec
pulvinar, porttitor tellus. Aliquam venenatis. Donec facilisis
pharetra tortor. In nec mauris eget magna consequat
convalis. Nam cras vitae mi vitae odio pellentesque interdum. Sed
consequat viverra nisl. Suspendisse arcu metus, blandit quis,
rhoncus ac, pharetra eget, velit. Mauris urna. Morbi nonummy
molestie orci. Praesent nisi elit, fringilla ac, suscipit non,
tristique vel, mauris. Curabitur vel lorem id nisl porta
adipiscing. Duis vulputate tristique enim. Donec quis lectus a
justo imperdiet tempus.  Suspendisse eu lectus. In nunc."""

text2_lines = text2.splitlines()
```

Comparing Bodies of Text

The `Differ` class works on sequences of text lines and produces human-readable *deltas*, or change instructions, including differences within individual lines. The default output produced by `Differ` is similar to the `diff` command-line tool under Unix. It includes the original input values from both lists, including common values, and markup data to indicate which changes were made.

- Lines prefixed with `-` were in the first sequence, but not the second.
- Lines prefixed with `+` were in the second sequence, but not the first.
- If a line has an incremental difference between versions, an extra line prefixed with `?` is used to highlight the change within the new version.
- If a line has not changed, it is printed with an extra blank space on the left column so that it is aligned with the other output that may have differences.

Breaking the text up into a sequence of individual lines before passing it to `compare()` produces more readable output than passing in large strings.

```
# difflib_differ.py

import difflib
from difflib_data import *

d = difflib.Differ()
diff = d.compare(text1_lines, text2_lines)
print('\n'.join(diff))
```

```
print('\n'.join(diff))
```

The beginning of both text segments in the sample data is the same, so the first line is printed without any extra annotation.

```
  Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit. Integer eu lacus accumsan arcu fermentum euismod. Donec
```

The third line of the data has been changed to include a comma in the modified text. Both versions of the line are printed, with the extra information on line 5 showing the column where the text was modified, including the fact that the `,` character was added.

```
- pulvinar porttitor tellus. Aliquam venenatis. Donec facilisis
+ pulvinar, porttitor tellus. Aliquam venenatis. Donec facilisis
?         +
```

The next few lines of the output show that an extra space was removed.

```
- pharetra tortor.  In nec mauris eget magna consequat
?                 -

+ pharetra tortor. In nec mauris eget magna consequat
```

Next, a more complex change was made, replacing several words in a phrase.

```
- convalis. Nam sed sem vitae odio pellentesque interdum. Sed
?                 - --

+ convalis. Nam cras vitae mi vitae odio pellentesque interdum. Sed
?               +++ +++++   +
```

The last sentence in the paragraph was changed significantly, so the difference is represented by removing the old version and adding the new.

```
  consequat viverra nisl. Suspendisse arcu metus, blandit quis,
  rhoncus ac, pharetra eget, velit. Mauris urna. Morbi nonummy
  molestie orci. Praesent nisi elit, fringilla ac, suscipit non,
  tristique vel, mauris. Curabitur vel lorem id nisl porta
- adipiscing. Suspendisse eu lectus. In nunc. Duis vulputate
- tristique enim. Donec quis lectus a justo imperdiet tempus.
+ adipiscing. Duis vulputate tristique enim. Donec quis lectus a
+ justo imperdiet tempus.  Suspendisse eu lectus. In nunc.
```

The `ndiff()` function produces essentially the same output. The processing is specifically tailored for working with text data and eliminating “noise” in the input.

Other Output Formats

While the `Differ` class shows all of the input lines, a *unified diff* includes only the modified lines and a bit of context. The `unified_diff()` function produces this sort of output.

```
# difflib_unified.py

import difflib
from difflib_data import *

diff = difflib.unified_diff(
    text1_lines,
    text2_lines,
    lineterm='',
)
print('\n'.join(diff))
```

The `lineterm` argument is used to tell `unified_diff()` to skip appending newlines to the control lines that it returns because the input lines do not include them. Newlines are added to all of the lines when they are printed. The output should look familiar to users of many popular version-control tools.

```
$ python3 difflib_unified.py
```

```
---
+++
@@ -1,11 +1,11 @@
 Lorem ipsum dolor sit amet, consectetuer adipiscing
 elit. Integer eu lacus accumsan arcu fermentum euismod. Donec
-pulvinar porttitor tellus. Aliquam venenatis. Donec facilisis
-pharetra tortor.  In nec mauris eget magna consequat
-convalis. Nam sed sem vitae odio pellentesque interdum. Sed
+pulvinar, porttitor tellus. Aliquam venenatis. Donec facilisis
+pharetra tortor. In nec mauris eget magna consequat
+convalis. Nam cras vitae mi vitae odio pellentesque interdum. S
ed
 consequat viverra nisl. Suspendisse arcu metus, blandit quis,
 rhoncus ac, pharetra eget, velit. Mauris urna. Morbi nonummy
 molestie orci. Praesent nisi elit, fringilla ac, suscipit non,
 tristique vel, mauris. Curabitur vel lorem id nisl porta
-adipiscing. Suspendisse eu lectus. In nunc. Duis vulputate
-tristique enim. Donec quis lectus a justo imperdiet tempus.
+adipiscing. Duis vulputate tristique enim. Donec quis lectus a
+justo imperdiet tempus.  Suspendisse eu lectus. In nunc.
```

Using `context_diff()` produces similar readable output.

Junk Data

All of the functions that produce difference sequences accept arguments to indicate which lines should be ignored and which characters within a line should be ignored. These parameters can be used to skip over markup or whitespace changes in two versions of a file, for example.

```
# difflib_junk.py

# This example is adapted from the source for difflib.py.

from difflib import SequenceMatcher


def show_results(match):
    print('  a    = {}'.format(match.a))
    print('  b    = {}'.format(match.b))
    print('  size = {}'.format(match.size))
    i, j, k = match
    print('  A[a:a+size] = {!r}'.format(A[i:i + k]))
    print('  B[b:b+size] = {!r}'.format(B[j:j + k]))


A = " abcd"
B = "abcd abcd"

print('A = {!r}'.format(A))
print('B = {!r}'.format(B))

print('\nWithout junk detection:')
s1 = SequenceMatcher(None, A, B)
match1 = s1.find_longest_match(0, len(A), 0, len(B))
show_results(match1)

print('\nTreat spaces as junk:')
s2 = SequenceMatcher(lambda x: x == " ", A, B)
match2 = s2.find_longest_match(0, len(A), 0, len(B))
show_results(match2)
```

The default for `Differ` is to not ignore any lines or characters explicitly, but rather to rely on the ability of `SequenceMatcher` to detect noise. The default for `ndiff()` is to ignore space and tab characters.

```
$ python3 difflib_junk.py

A = ' abcd'
B = 'abcd abcd'
```

```
Without junk detection:
  a    = 0
  b    = 4
  size = 5
  A[a:a+size] = ' abcd'
  B[b:b+size] = ' abcd'

Treat spaces as junk:
  a    = 1
  b    = 0
  size = 4
  A[a:a+size] = 'abcd'
  B[b:b+size] = 'abcd'
```

Comparing Arbitrary Types

The SequenceMatcher class compares two sequences of any types, as long as the values are hashable. It uses an algorithm to identify the longest contiguous matching blocks from the sequences, eliminating “junk” values that do not contribute to the real data.

The funct get_opcodes() returns a list of instructions for modifying the first sequence to make it match the second. The instructions are encoded as five-element tuples, including a string instruction (the “opcode”, see the table below) and two pairs of start and stop indexes into the sequences (denoted as i1, i2, j1, and j2).

difflib.get_opcodes() Instructions

Opcode	Definition
'replace'	Replace a[i1:i2] with b[j1:j2]
'delete'	Remove a[i1:i2] entirely
'insert'	Insert b[j1:j2] at a[i1:i1]
'equal'	The subsequences are already equal

```
# difflib_seq.py

import difflib

s1 = [1, 2, 3, 5, 6, 4]
s2 = [2, 3, 5, 4, 6, 1]

print('Initial data:')
print('s1 =', s1)
print('s2 =', s2)
print('s1 == s2:', s1 == s2)
print()

matcher = difflib.SequenceMatcher(None, s1, s2)
for tag, i1, i2, j1, j2 in reversed(matcher.get_opcodes()):

    if tag == 'delete':
        print('Remove {} from positions [{}:{}]'.format(
            s1[i1:i2], i1, i2))
        print('  before =', s1)
        del s1[i1:i2]

    elif tag == 'equal':
        print('s1[{}:{}] and s2[{}:{}] are the same'.format(
            i1, i2, j1, j2))

    elif tag == 'insert':
        print('Insert {} from s2[{}:{}] into s1 at {}'.format(
            s2[j1:j2], j1, j2, i1))
        print('  before =', s1)
        s1[i1:i2] = s2[j1:j2]

    elif tag == 'replace':
        print(('Replace {} from s1[{}:{}] '
               'with {} from s2[{}:{}]').format(
```

```
                    s1[i1:i2], i1, i2, s2[j1:j2], j1, j2))
            print('  before =', s1)
            s1[i1:i2] = s2[j1:j2]

        print('   after =', s1, '\n')

print('s1 == s2:', s1 == s2)
```

This example compares two lists of integers and uses `get_opcodes()` to derive the instructions for converting the original list into the newer version. The modifications are applied in reverse order so that the list indexes remain accurate after items are added and removed.

```
$ python3 difflib_seq.py

Initial data:
s1 = [1, 2, 3, 5, 6, 4]
s2 = [2, 3, 5, 4, 6, 1]
s1 == s2: False

Replace [4] from s1[5:6] with [1] from s2[5:6]
  before = [1, 2, 3, 5, 6, 4]
   after = [1, 2, 3, 5, 6, 1]

s1[4:5] and s2[4:5] are the same
   after = [1, 2, 3, 5, 6, 1]

Insert [4] from s2[3:4] into s1 at 4
  before = [1, 2, 3, 5, 6, 1]
   after = [1, 2, 3, 5, 4, 6, 1]

s1[1:4] and s2[0:3] are the same
   after = [1, 2, 3, 5, 4, 6, 1]

Remove [1] from positions [0:1]
  before = [1, 2, 3, 5, 4, 6, 1]
   after = [2, 3, 5, 4, 6, 1]

s1 == s2: True
```

`SequenceMatcher` works with custom classes, as well as built-in types, as long as they are hashable.

See also

- Standard library documentation for difflib
- "Pattern Matching: The Gestalt Approach" – Discussion of a similar algorithm by John W. Ratcliff and D. E. Metzener published in *Dr. Dobb's Journal* in July, 1988.

re — Regular Expressions

Data Structures

Quick Links
Comparing Bodies of Text
Other Output Formats
Junk Data
Comparing Arbitrary Types

This page was last updated 2017-01-28.

Navigation
re — Regular Expressions
Data Structures

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example