Модуль difflib в Python, сравнение текста

Ста... / Модуль difflib в Python, сравнение текста

Ср...ычисление различий

Исх.../difflib.py

Мод... и функции для сравнения последовательностей. Модуль может быть использован, для сравнения тек...сы и функции, которые создают отчеты в нескольких распространенных форматах, включая HTML. Для ...лов смотрите модуль filecmp.

В примерах будем использовать следующие два текста:

```
txt1 = """Модуль difflib в основном используется, для сравнения
текстовых последовательностей и включает в себя классы и функции,
которые создают отчеты в нескольких распространенных форматов,
включая HTML."""

txt2 = """Модуль `difflib` может быть использован, для сравнения
текстовых последовательностей и включает в себя функции,
которые создают отчеты в нескольких распространенных форматах,
включая HTML."""
```

Сравнение текстовых строк.

Класс difflib.Differ() работает с последовательностями строк текста и производит удобочитаемые дельты или инструкцию по изменению и выводит различия в отдельных строках. Класс Differ(), похож на инструмент командной строки diff в Unix системах. Он включает исходные входные значения из обоих списков строк текста, включая общие значения и данные разметки, чтобы указать какие изменения были внесены.

- Строки с префиксом - присутствуют в первой последовательности, но не во второй.
- Строки с префиксом + присутствуют во второй последовательности, но не в первой.
- Если строка имеет инкрементную разницу между версиями, дополнительная строка с префиксом ? используется для выделения изменений в новой версии.
- Если строка не изменилась, она печатается с лишним пробелом в левом столбце, чтобы она выровнялась с другими выходными данными, которые могут иметь различия.

Разбиение текста на последовательность отдельных строк перед его передачей в метод compare() класса Differ() приводит к более читабельному выводу, чем передача больших строк.

```
import difflib as df

# Перед тем как сравнивать, разобьем
# тексты на строки
txt1_list = txt1.splitlines()
txt2_list = txt2.splitlines()

d = df.Differ()
diff = d.compare(txt1_list, txt2_list)
print('\n'.join(diff))
# - Модуль difflib в основном используется, для сравнения
# + Модуль `difflib` может быть использован, для сравнения
# - текстовых последовательностей и включает в себя классы и функции,
# ?                                                 ---------
#
# ...товых последовательностей и включает в себя функции,
# ...ые создают отчеты в нескольких распространенных форматов,
```

```
# ?                                                    ^^
#
# + которые создают отчеты в нескольких распространенных форматах,
# ?                                                    ^^
#   включая HTML.
```


Функция ndiff() модуля difflib производит же результат.

```
import difflib as df

txt1_list = txt1.splitlines()
txt2_list = txt2.splitlines()

# выдаст тот же результат
diff = df.ndiff(txt1_list, txt2_list)
print('\n'.join(diff))
```

В то время как класс difflib.Differ() показывает все входные строки, унифицированная функция unified_diff() содержит только измененные строки и немного контекста.

```
import difflib as df

txt1_list = txt1.splitlines()
txt2_list = txt2.splitlines()

diff = df.unified_diff(txt1_list, txt2_list, lineterm='')
print('\n'.join(diff))
# ---
# +++
# @@ -1,4 +1,4 @@
# -Модуль difflib в основном используется, для сравнения
# -текстовых последовательностей и включает в себя классы и функции,
# -которые создают отчеты в нескольких распространенных форматов,
# +Модуль `difflib` может быть использован, для сравнения
# +текстовых последовательностей и включает в себя функции,
# +которые создают отчеты в нескольких распространенных форматах,
#  включая HTML.
```

Аргумент lineterm указывает функции unified_diff() о пропуске пустых строк, которые она возвращает. Новые строки добавляются ко всем строкам при их печати, для того что бы вывод был похож на популярные инструменты контроля версий.

Использование функции context_diff(), так же дает читаемый вывод.

Игнорирование строк и символов в анализе.

Все функции, которые создают разностные последовательности, принимают аргументы, указывающие, какие строки следует игнорировать и какие символы в строке следует игнорировать. Эти параметры можно использовать, например, для пропуска изменений разметки или пробелов в двух версиях файла.

```
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('\nБез обнаружения мусора:')
s1 = SequenceMatcher(None, A, B)
match1 = s1.find_longest_match(0, len(A), 0, len(B))
show_results(match1)

print('\nРассматривает пробелы как мусор:')
```

```
s2 = SequenceMatcher(lambda x: x == ' ', A, B)
match2 = s2.find_longest_match(0, len(A), 0, len(B))
show_results(match2)
```

По умолчанию класс Differ() не игнорирует никакие строки или символы явно, а скорее полагается на способность класса SequenceMatcher() обнаруживать шум. По умолчанию для функция difflib.ndiff() игнорируется пробел и символы табуляции.


```
# Получим вывод:

A = ' abcd'
B = 'abcd abcd'

Без обнаружения мусора:
  a    = 0
  b    = 4
  size = 5
  A[a:a+size] = ' abcd'
  B[b:b+size] = ' abcd'

Рассматривает пробелы как мусор:
  a    = 1
  b    = 0
  size = 4
  A[a:a+size] = 'abcd'
  B[b:b+size] = 'abcd'
```

Сравнение последовательностей произвольных типов.

Класс SequenceMatcher() сравнивает две последовательности любых видов, если их элементы являются хешируемыми. Он использует алгоритм для идентификации наиболее длинных непрерывных блоков соответствия из последовательностей, устраняя "мусорные" значения, которые не вносят вклад в реальные данные.

Функция get_opcodes() класса SequenceMatcher() возвращает список инструкций для изменения первой последовательности, чтобы она соответствовала второй. Инструкции кодируются в виде пятиэлементных кортежей, включая строковую инструкцию и две пары индексов start и stop в последовательности, обозначаемые как i1, i2 и j1, j2.

В этом примере сравниваются два списка целых чисел, в котором используется метод get_opcodes() для получения инструкций по преобразованию исходного списка в более новую версию. Изменения применяются в обратном порядке, поэтому индексы списка остаются верными после добавления и удаления элементов.

```
import difflib as df

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

print('s1 =', s1)
print('s2 =', s2)
print('s1 == s2:', s1 == s2, '\n')

matcher = df.SequenceMatcher(None, s1, s2)
for tag, i1, i2, j1, j2 in reversed(matcher.get_opcodes()):
    if tag == 'delete':
        print(f'Удалить {s1[i1:i2]} из позиции [{i1}:{i2}]')
        print('до =\t', s1)
        del s1[i1:i2]
    elif tag == 'equal':
        print(f's1[{i1}:{i2}] и s2[{j1}:{j2}] одинаковы')
    elif tag == 'insert':
        print(f'Вставить {s2[j1:j2]} из s2[{j1}:{j2}] в s1 перед {s1[i1]}')
        print('до =\t', s1)
        s1[i1:i2] = s2[j1:j2]
    elif tag == 'replace':
        print(f'Заменить {s1[i1:i2]} из s1[{i1}:{i2}] на {s2[j1:j2]} из s2[{j1}:{j2}]')
        print('до =\t', s1)
        s1[i1:i2] = s2[j1:j2]

    print('после =\t', s1, '\n')

pr == s2:', s1 == s2)
```

SequenceMatcher() работает с пользовательскими классами, а также со встроенными типами, если их элементы являются хешируемыми.

Получим вывод:


```
s1 == s2: False

Заменить [4] из s1[5:6] на [1] из s2[5:6]
до =     [1, 2, 3, 5, 6, 4]
после =  [1, 2, 3, 5, 6, 1]

s1[4:5] и s2[4:5] одинаковы
после =  [1, 2, 3, 5, 6, 1]

Вставить [4] из s2[3:4] в s1 перед 6
до =     [1, 2, 3, 5, 6, 1]
после =  [1, 2, 3, 5, 4, 6, 1]

s1[1:4] и s2[0:3] одинаковы
после =  [1, 2, 3, 5, 4, 6, 1]

Удалить [1] из позиции [0:1]
до =     [1, 2, 3, 5, 4, 6, 1]
после =  [2, 3, 5, 4, 6, 1]

s1 == s2: True
```

Содержание раздела: