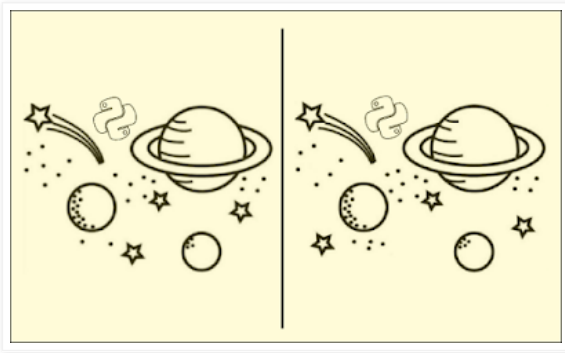
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 20 de mayo de 2016

Difflib: encontrando las diferencias



El módulo **difflib** proporciona clases y funciones para comparar secuencias (cadenas de texto, listas y tuplas) y obtener una salida con las diferencias encontradas, en distintos formatos. Resulta de utilidad para comparar el contenido de dos archivos.

Clase Differ. Comparando secuencias

La clase **Differ** se utiliza para comparar las líneas de dos secuencias y obtener las diferencias encontradas en un formato legible (deltas). El resultado que devuelve es similar al que ofrece el comando **diff** de comparación de archivos de Unix y GNU/Linux, que utiliza una serie de caracteres para destacar las diferencias encontradas:

- El carácter '**-**' al comienzo de una línea indica que dicha línea aparece en la primera secuencia, pero no en la segunda.
- El carácter '**+**' al comienzo de una línea indica que dicha línea aparece en la segunda secuencia, pero no en la primera.
- El carácter de espacio en blanco al comienzo de una línea indica que dicha línea es coincidente en las dos secuencias.
- El carácter '?' al comienzo de una línea muestra con cierto detalle las diferencias que hay en las líneas no coincidentes.

En las líneas con '**?**' se utilizan los caracteres '**+**', '**-**' y '**^**' para marcar los elementos que son distintos:

- El carácter '**-**' es para señalar aquellos caracteres que aparecen en la primera secuencia, pero no en la segunda;
- el carácter '**+**' para señalar aquellos que aparecen en la secuencia secuencia y no en la primera;
- y el carácter '**^**' para destacar los caracteres diferentes tanto en la primera como en la segunda secuencia.

Estos caracteres son colocados estratégicamente para facilitar la lectura de la salida.

La clase **Differ** utiliza la clase **SequenceMatcher** tanto para comparar las secuencias de líneas como las secuencias de caracteres.

Obtener diferencias entre textos con Differ() y compare()

En el siguiente ejemplo se comparan dos textos con **Differ()** y **compare()**:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import difflib
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

```
# Declara variables con dos textos con ciertas diferencias

texto1 = """Antigua Roma:
Antigua Roma (en latín, Antiqua Rōma) designa la
entidad política unitaria surgida de la expansión de la ciudad
de Roma, que en su época de apogeo, llegó a abarcar desde Gran
Bretaña al desierto del Sahara y desde la península ibérica al
Éufrates. En un principio, tras su fundación (según la
tradición en 753 aC), Roma fue una monarquía etrusca. Más
tarde (509 aC) fue una república latina, y en 27 aC
se convirtió en un imperio."""

texto2 = """Antigua Roma (en latín, Antiqua Rōma) designa la
entidad política unitaria surgida de la expansión de la ciudad
de Roma, que en su época de apogeo, llegó a abarcar desde Gran
Bretaña al desierto del Sahara y desde la península ibérica al
Éufrates. En un principio, tras su creación (según la
tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
tarde (509 a. C.) fue una república latina, y en 27 a. C.
se convirtió en un imperio"""

# Crea listas con las líneas de los textos anteriores

lineas1 = texto1.splitlines()
lineas2 = texto2.splitlines()

# Obtiene diferencias

diferencia = difflib.Differ()
generador_diferencia = diferencia.compare(lineas1, lineas2)
print('lineas1', type(lineas1), 'lineas2', type(lineas2))

print('\n'.join(generador_diferencia))

'''
Salida
------

- Antigua Roma:
  Antigua Roma (en latín, Antiqua Rōma) designa la
  entidad política unitaria surgida de la expansión de la ciudad
  de Roma, que en su época de apogeo, llegó a abarcar desde Gran
  Bretaña al desierto del Sahara y desde la península ibérica al
- Éufrates. En un principio, tras su fundación (según la
?                                    ^^^^

+ Éufrates. En un principio, tras su creación (según la
?                                    ^^^

- tradición en 753 aC), Roma fue una monarquía etrusca. Más
+ tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
?                  ++ +

- tarde (509 aC) fue una república latina, y en 27 aC
+ tarde (509 a. C.) fue una república latina, y en 27 a. C.
?             ++ +                                   ++ +

- se convirtió en un imperio.
?                            -

+ se convirtió en un imperio
'''
```

Obtener diferencias entre textos con ndiff()

La función **ndiff()** produce, en principio, la misma salida que **compare()** pero su procesamiento está diseñado específicamente para trabajar con cadenas y permite priorizar unas comparaciones sobre otras si se cumplen determinadas condiciones, utilizando los argumentos opcionales **linejunk** (línea basura) y/o **charjunk** (caracter basura).

difflib.ndiff(a, b, linejunk=None, charjunk=IS_CHARACTER_JUNK**)**

En el siguiente ejemplo se comparan los dos textos anteriores utilizando **ndiff()** y el resultado que se obtiene es exactamente el mismo que con **compare()**.

```
generador_diferencia = difflib.ndiff(lineas1, lineas2)
print('\n'.join(generador_diferencia))

'''
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

mayo 2016 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
Salida
------

- Antigua Roma:
  Antigua Roma (en latín, Antiqua Rōma) designa la
  entidad política unitaria surgida de la expansión de la ciudad
  de Roma, que en su época de apogeo, llegó a abarcar desde Gran
  Bretaña al desierto del Sahara y desde la península ibérica al
- Éufrates. En un principio, tras su fundación (según la
?                                    ^^^^

+ Éufrates. En un principio, tras su creación (según la
?                                    ^^^

- tradición en 753 aC), Roma fue una monarquía etrusca. Más
+ tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
?                   ++ +

- tarde (509 aC) fue una república latina, y en 27 aC
+ tarde (509 a. C.) fue una república latina, y en 27 a. C.
?             ++ +                                      ++ +

- se convirtió en un imperio.
?                            -

+ se convirtió en un imperio
'''
```

Ajustar los resultados de ndiff() con linejunk y charjunk

Los argumentos **linejunk** (línea basura) y **charjunk** (carácter basura) se utilizan para modificar la salida predeterminada priorizando unas comparaciones sobre otras en función a una necesidad concreta.

- **linejunk** puede tener el valor por defecto **None** o el valor devuelto por una función (de un sólo argumento que se corresponde con la línea a comparar en un momento dado). Dicha línea puede analizarse dentro de la función y dependiendo de su contenido hacer que la función devuelva **True** o **False**. Si el valor devuelto es **True** la línea tendría una prioridad más baja a la hora de compararse que otra similar que devuelva **False**. El módulo cuenta con una función llamada **IS_LINE_JUNK()** que se utiliza para ignorar líneas en blanco o que contienen un solo carácter '**#**'.
- **charjunk** tiene el mismo propósito que **linejunk** pero a nivel de carácter. Por defecto tiene el valor **True** o **False** que devuelve la función de módulo **IS_CHARACTER_JUNK**. Cuando dicha función devuelve el valor **True** entonces el carácter analizado se corresponde con un espacio en blanco o un tabulador. También, **charjunk** puede obtener el valor de una función hecha a medida de la misma forma que con **linejunk**.

En el siguiente ejemplo el resultado de la comparación que se obtiene con **compare()** y **ndiff()** es diferente. La función **ndiff()** incluye el argumento **charjunk** que recibe el valor que devuelve una función **lambda**. Esta función analiza caracteres para modificar el comportamiento predeterminado de **ndiff()**. Concretamente, ante situaciones de igualdad prioriza comparaciones de cadenas que no contengan el carácter 'o'. La función **lambda** devuelve **True** o **False** dependiendo de si se detecta o no dicho carácter.

```
lista1 = ["uno\n", "dos\n", "tres\n"]
lista2 = ["uno\n", "osd\n", "dss\n", "tres\n"]

# Obtener diferencias con Differ() y compare():

generador_diferencia1 = difflib.Differ().compare(lista1, lista2)
print(''.join(generador_diferencia1))

'''
Salida
------

  uno
- dos
? -
+ osd
?   +
+ dss
  tres
'''

# Obtener diferencias con ndiff():

generador_diferencia2 = difflib.ndiff(lista1, lista2,
```

```
                                    charjunk=lambda x: x=='o')
print(''.join(generador_diferencia2))

'''
Salida
------

  uno
+ osd
- dos
?  ^
+ dss
?  ^
  tres
'''
```

Obtener diferencias con restore()

La función **restore()** devuelve del resultado obtenido en una comparación con **compare()** o **ndiff()** todas las líneas coincidentes y, de una de las dos secuencias, aquellas que tengan diferencias; suprimiendo las marcas iniciales en cada línea.

difflib.restore(sequence, which**)**

El argumento **which** fija con el valor 1 o 2 la secuencia a devolver.

A continuación, un ejemplo para obtener las líneas comunes y aquellas que son distintas en la segunda secuencia:

```
generador_diferencia = difflib.ndiff(lineas1, lineas2)
print('\n'.join(difflib.restore(generador_diferencia, which=2)))

'''
Salida
------

Antigua Roma (en latín, Antiqua Rōma) designa la
entidad política unitaria surgida de la expansión de la ciudad
de Roma, que en su época de apogeo, llegó a abarcar desde Gran
Bretaña al desierto del Sahara y desde la península ibérica al
Éufrates. En un principio, tras su creación (según la
tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
tarde (509 a. C.) fue una república latina, y en 27 a. C.
se convirtió en un imperio
'''
```

Obtener diferencias unificadas con unified_diff()

Mientras que con la clase **Differ** se muestran todas las líneas de entrada, con **unified_diff()** (diferencias unificadas) sólo se incluyen algunas líneas (por defecto, 3) que no tienen diferencias para "situar" el contexto y las líneas que tienen diferencias, utilizando un formato más compacto.

difflib.unified_diff(a, b, fromfile='', tofile='', fromfiledate='', tofiledate='', n=3, lineterm='\n'**)**

Ejemplo:

```
texto1 = """Antigua Roma:
Antigua Roma (en latín, Antiqua Rōma) designa la
entidad política unitaria surgida de la expansión de la ciudad
de Roma, que en su época de apogeo, llegó a abarcar desde Gran
Bretaña al desierto del Sahara y desde la península ibérica al
Éufrates. En un principio, tras su fundación (según la
tradición en 753 aC), Roma fue una monarquía etrusca. Más
tarde (509 aC) fue una república latina, y en 27 aC
se convirtió en un imperio.
Al período de mayor esplendor se le conoce como pax romana,
debido al relativo estado de armonía que prevaleció en las
regiones que estaban bajo el dominio romano, un período de
orden y prosperidad que conoció el Imperio bajo la Dinastía de
los Antoninos (96-192) y, en menor medida, bajo la de los
Severos (193-235). Constituye un elemento crucial del
desarrollo de Occidente, y más tarde también de Oriente.
"""

texto2 = """Antigua Roma (en latín, Antiqua Rōma) designa la
entidad política unitaria surgida de la expansión de la ciudad
de Roma, que en su época de apogeo, llegó a abarcar desde Gran
Bretaña al desierto del Sahara y desde la península ibérica al
```

```
Éufrates. En un principio, tras su creación (según la
tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
tarde (509 a. C.) fue una república latina, y en 27 a. C.
se convirtió en un imperio
Al período de mayor esplendor se le conoce como pax romana,
debido al relativo estado de armonía que prevaleció en las
regiones que estaban bajo el dominio romano, un período de
orden y prosperidad que conoció el Imperio bajo la Dinastía de
los Antoninos (96-192) y, en menor medida, bajo la de los
Severos (193-235). Constituye un elemento crucial del
desarrollo de Occidente, y más tarde también de Oriente.
"""

lineas1 = texto1.splitlines()
lineas2 = texto2.splitlines()

generador_diferencia = difflib.unified_diff(lineas1,
                                            lineas2, lineterm='',)
print('\n'.join(list(generador_diferencia)))

'''
Salida
------

---
+++
@@ -1,12 +1,11 @@
-Antigua Roma:
 Antigua Roma (en latín, Antiqua Rōma) designa la
 entidad política unitaria surgida de la expansión de la ciudad
 de Roma, que en su época de apogeo, llegó a abarcar desde Gran
 Bretaña al desierto del Sahara y desde la península ibérica al
-Éufrates. En un principio, tras su fundación (según la
-tradición en 753 aC), Roma fue una monarquía etrusca. Más
-tarde (509 aC) fue una república latina, y en 27 aC
-se convirtió en un imperio.
+Éufrates. En un principio, tras su creación (según la
+tradición en 753 a. C.), Roma fue una monarquía etrusca. Más
+tarde (509 a. C.) fue una república latina, y en 27 a. C.
+se convirtió en un imperio
 Al período de mayor esplendor se le conoce como pax romana,
 debido al relativo estado de armonía que prevaleció en las
 regiones que estaban bajo el dominio romano, un período
'''
```

La función **context_diff()** reproduce una salida similar.

Obtener las mejores coincidencias con get_close_matches()

La función **get_close_matches()** devuelve una lista con las consideradas mejores coincidencias de una lista de posibilidades.

difflib.get_close_matches(word, possibilities, n=3, cutoff=0.6**)**

El argumento **n** define el número máximo de coincidencias a devolver y el argumento **cutoff**, que tendrá un valor entre 0 y 1, establece el ratio o índice de coincidencia exigido.

El siguiente ejemplo muestra la palabra más próxima que tenga un índice, al menos, del 80%

```
lista = 'moto'
posibilidades = ['ciclomotor', 'motero', 'bicicleta',
                 'motorista', 'motor']

print(difflib.get_close_matches(lista, posibilidades,
                                n=1, cutoff=0.8))  # motor

'''
Salida
------

['motor']
'''
```

Obtener diferencias entre códigos HTML con HtmlDiff()

Para obtener las diferencias entre dos códigos **HTML** se pueden utilizar los métodos **make_table()** y **make_file()**.

El método **make_table()** compara dos listas de cadenas que contienen líneas de código **HTML** y

devuelve una tabla **HTML** con las diferencias encontradas:

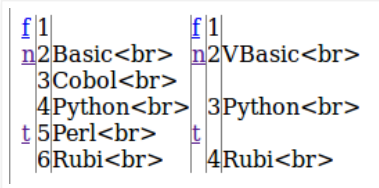
make_table(fromlines, tolines, fromdesc='', todesc='', context=False, numlines=5**)**

```
encabezado = '<html><head><title>Lenguajes</title></head><body>'
pie = '</body></html>'

html1 = """
Basic<br>
Cobol<br>
Python<br>
Perl<br>
Rubi<br>
"""
html2 = """
VBasic<br>
Python<br>
Rubi<br>
"""

lineas1 = html1.splitlines()
lineas2 = html2.splitlines()

diferencia_html = difflib.HtmlDiff()
tabla_html = diferencia_html.make_table(lineas1, lineas2)
pagina = encabezado + tabla_html + pie
print(pagina)
```

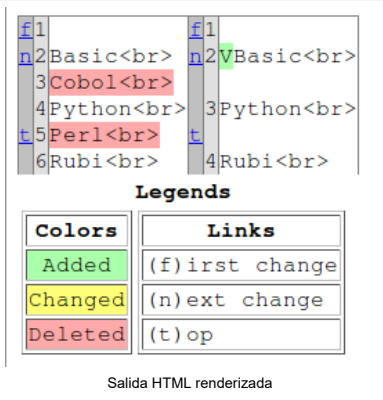


Salida HTML renderizada

El método **make_file()** compara dos listas de cadenas que contienen líneas de código **HTML** y devuelve un archivo **HTML** con las diferencias encontradas:

make_file(fromlines, tolines, fromdesc='', todesc='', context=False, numlines=5, *, charset='utf-8'**)**

```
archivo_html = diferencia_html.make_file(lineas1, lineas2)
print(archivo_html)
```



Salida HTML renderizada

La clase difflib.SequenceMatcher

SequenceMatcher es una clase flexible para comparar pares de secuencias de cualquier tipo (siempre que los elementos de la secuencia sean objetos *hashable*, es decir, que tengan un identificador '*hash*' que no cambia durante la vida útil de un objeto y que permite comparar con otros objetos).

La clase **Differ** utiliza la clase **SequenceMatcher** tanto para comparar las secuencias de líneas como de caracteres (dentro de aquellas líneas que sean similares).

class difflib.SequenceMatcher(isjunk=None, a='', b='', autojunk=True**)**

SequenceMatcher trata de forma automática como '*basura*' ciertos elementos de una secuencia. También, comprueba las veces que aparece cada elemento de forma individual y, si las veces que aparece un elemento después de un primer análisis representan más del 1% en una secuencia con al menos 200 elementos, dicho elemento es marcado como '*popular*' y tratado

como '*basura*'. Esto se hace para facilitar las coincidencias entre dos secuencias que se comparan. De todas formas, este modo de trabajar se puede deshabilitar asignando **False** al argumento **autojunk** en el momento de definir un objeto **SequenceMatcher**.

El argumento **autojunk** está disponible desde Python 3.2

El argumento opcional **injunk** tiene el valor por defecto **None** o el valor devuelto por una función que toma un elemento de la secuencia, lo analiza y devuelve **True** si dicho elemento es '*basura*', para que este sea ignorado. El valor **None** es equivalente a la función *lambda* x: 0 que no tiene ningún efecto cuando no hay elementos.

Métodos para establecer secuencias a comparar

- **set_seqs(**a, b): Establece las dos secuencias a comparar
- **set_seq1(**a): Establece la primera secuencia a comparar
- **set_seq2(**b): Establece la segunda secuencia a comparar

Obtener nivel de coincidencias (ratio) entre dos cadenas

El método **ratio()** devuelve un valor entre 0 y 1 que representa el ratio o nivel de coincidencias entre dos secuencias.

```
c1 = "en un\t lugar de la Mancha"
c2 = "un lugar de\t la galaxia"
c3 = "un lugar de la Galaxia"

dif1 = difflib.SequenceMatcher(lambda x: x == " \t", c1, c2)
dif2 = difflib.SequenceMatcher(lambda x: x == " \t", c2, c3)
print(dif1.ratio())  # 0.7083333333333334
print(dif2.ratio())  # 0.9333333333333333
```

Otras opciones más rápidas: **quick_ratio()** y **real_quick_ratio()**.

Obtener tuplas que describen los pasos para convertir a en b

El método **get_opcodes()** devuelve una lista de tuplas que describen los pasos que habría que dar para convertir la secuencia **a** en **b**.

```
for tuplas1 in dif1.get_opcodes():
    print(tuplas1)

print()
for tuplas2 in dif2.get_opcodes():
    print(tuplas2)

'''
Salida
------

('delete', 0, 3, 0, 0)
('equal', 3, 5, 0, 2)
('delete', 5, 6, 2, 2)
('equal', 6, 15, 2, 11)
('insert', 15, 15, 11, 12)
('equal', 15, 19, 12, 16)
('replace', 19, 20, 16, 17)
('equal', 20, 21, 17, 18)
('replace', 21, 24, 18, 19)
('equal', 24, 25, 19, 20)
('insert', 25, 25, 20, 23)

('equal', 0, 11, 0, 11)
('delete', 11, 12, 11, 11)
('equal', 12, 16, 11, 15)
('replace', 16, 17, 15, 16)
('equal', 17, 23, 16, 22)
'''
```

Obtener las posiciones con las coincidencias de dos secuencias

El método **get_matching_blocks()** devuelve todas las posiciones con las coincidencias existentes entre dos secuencias:

```
for bloque1 in dif1.get_matching_blocks():
    print("c1[%d] y c2[%d] coinciden en %d elementos" % bloque1)
```

```
print()
for bloque2 in dif2.get_matching_blocks():
    print("c1[%d] y c3[%d] coinciden en %d elementos" % bloque2)

'''
Salida
------

c1[3] y c2[0] coinciden en 2 elementos
c1[6] y c2[2] coinciden en 9 elementos
c1[15] y c2[12] coinciden en 4 elementos
c1[20] y c2[17] coinciden en 1 elementos
c1[24] y c2[19] coinciden en 1 elementos
c1[25] y c2[23] coinciden en 0 elementos

c1[0] y c3[0] coinciden en 11 elementos
c1[12] y c3[11] coinciden en 4 elementos
c1[17] y c3[16] coinciden en 6 elementos
c1[23] y c3[22] coinciden en 0 elementos
'''
```

Relacionado:

- Filecmp: comparando archivos y directorios
- Explorando directorios con os.listdir, os.walk y os.scandir
- Filtrando archivos y directorios con glob y fnmatch
- Copiar, mover y borrar archivos/directorios con shutil

Ir al índice del tutorial de Python

Publicado por Pherkad en 17:21

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua