Optimizando Python usando Cython

Disclaimer

No soy ningun experto en el tema de usar Cython

Vean la charla de Facundo Batista:
Python más rápido que C

Hay varias fomas de usar Cython. Aca voy a mostrar una.

github.com/tzulberti/charlas

Levenshtein Distance

Calcula la cantidad de operaciones (cambiar un char por otro, sacar, o agregar uno) para convertir de una palabra a otra

1. ola y Hola => necesito agregar la H asique la distancia es 1

2. chau y char => necesito cambiar la **u** por la **r**, por lo tanto, la distancia también es 1

Levenshtein Distance

```
function LevenshteinDistance(char s[1..m], char t[1..n]):
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t
  // note that d has (m+1)*(n+1) values
  declare int d[0..m, 0..n]

  set each element in d to zero

  // source prefixes can be transformed into empty string by
  // dropping all characters
  for i from 1 to m:
      d[i, 0] := i

  // target prefixes can be reached from empty source prefix
  // by inserting every character
  for j from 1 to n:
      d[0, j] := j

  for j from 1 to n:
      for i from 1 to m:
          if s[i] = t[j]:
            substitutionCost := 0
          else:
            substitutionCost := 1
          d[i, j] := minimum(d[i-1, j] + 1,                   // deletion
                             d[i, j-1] + 1,                   // insertion
                             d[i-1, j-1] + substitutionCost)  // substitution

  return d[m, n]
```

Levenshtein Distance

```python
def levenshtein(seq1, seq2):
    size_x = len(seq1) + 1
    size_y = len(seq2) + 1
    matrix = [[0] * size_y for _ in range(size_x)]

    for x in range(size_x):
        matrix[x][0] = x
    for y in range(size_y):
        matrix[0][y] = y

    for x in range(1, size_x):
        for y in range(1, size_y):
            if seq1[x-1] == seq2[y-1]:
                substitution_cost = 0
            else:
                substitution_cost = 1

            matrix[x][y] = min(
                matrix[x-1][y] + 1,  # deletion
                matrix[x][y-1] + 1,  # insertion
                matrix[x-1][y-1] + substitution_cost, #substitution
            )
```

jampp

Codigo C

```c
#define MIN3(a, b, c) ((a) < (b) ? ((a) < (c) ? (a) : (c)) : ((b) < (c) ? (b) : (c)))

int levenshtein(char *s1, char *s2) {
    unsigned int x, y, s1len, s2len;
    s1len = strlen(s1);
    s2len = strlen(s2);
    unsigned int matrix[s2len+1][s1len+1];
    matrix[0][0] = 0;
    for (x = 1; x <= s2len; x++)
        matrix[x][0] = matrix[x-1][0] + 1;
    for (y = 1; y <= s1len; y++)
        matrix[0][y] = matrix[0][y-1] + 1;
    for (x = 1; x <= s2len; x++)
        for (y = 1; y <= s1len; y++)
            matrix[x][y] = MIN3(matrix[x-1][y] + 1, matrix[x][y-1] + 1, matrix[x-1][y-1] + (s1[y-1] == s2[x-1] ? 0 : 1));

    return(matrix[s2len][s1len]);
}
```

Benchmark

- Use este diccionario de palabras en ingles: https://raw.githubusercontent.com/dwyl/english-words/master/words.txt

- Convierto en un set cosa de que no necesariamente tenga el mismo orden

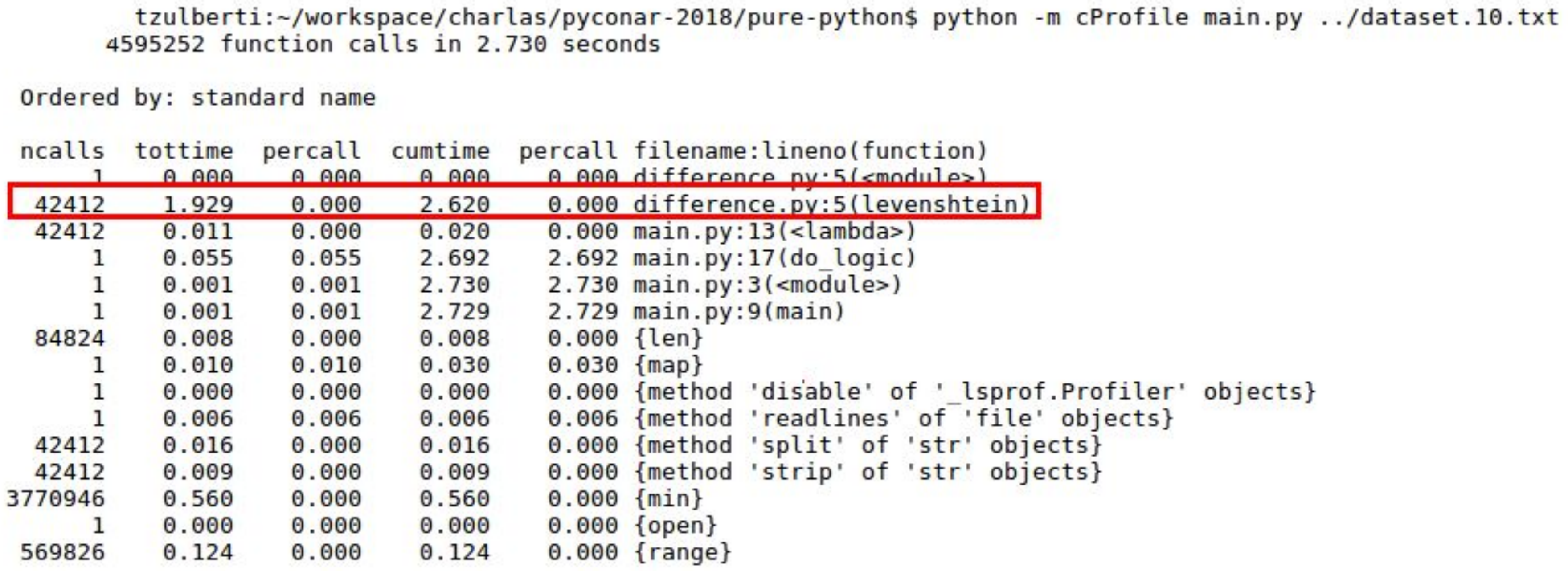
- Creo distintos archivos con diferentes cantidad de pares de palabras del set. Los distintos archivos además de tener diferente cantidad de palabras tienen diferentes pares de palabras

Comparación de performance

Cantidad de pares de palabras	C Puro	Python Puro	X veces más lento
233271	0.051	13.846	271
116635	0.025	7.785	311
58317	0.01P	3.506	318
29158	0.008	1.561	195

En promedio, Python es **273** más lento que C

Buscando el cuello de botella



```
         tzulberti:~/workspace/charlas/pyconar-2018/pure-python$ python -m cProfile main.py ../dataset.10.txt
         4595252 function calls in 2.730 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 difference.py:5(<module>)
    42412    1.929    0.000    2.620    0.000 difference.py:5(levenshtein)
    42412    0.011    0.000    0.020    0.000 main.py:13(<lambda>)
        1    0.055    0.055    2.692    2.692 main.py:17(do_logic)
        1    0.001    0.001    2.730    2.730 main.py:3(<module>)
        1    0.001    0.001    2.729    2.729 main.py:9(main)
    84824    0.008    0.000    0.008    0.000 {len}
        1    0.010    0.010    0.030    0.030 {map}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.006    0.006    0.006    0.006 {method 'readlines' of 'file' objects}
    42412    0.016    0.000    0.016    0.000 {method 'split' of 'str' objects}
    42412    0.009    0.000    0.009    0.000 {method 'strip' of 'str' objects}
  3770946    0.560    0.000    0.560    0.000 {min}
        1    0.000    0.000    0.000    0.000 {open}
   569826    0.124    0.000    0.124    0.000 {range}
```

Usando extensiones en C

```c
#include <Python.h>

static PyObject *
greet_name(PyObject *self, PyObject *args)
{
    const char *name;

    if (!PyArg_ParseTuple(args, "s", &name))
    {
        return NULL;
    }

    printf("Hello %s!\n", name);

    Py_RETURN_NONE;
}
```

```c
static PyMethodDef GreetMethods[] = {
    {"greet", greet_name, METH_VARARGS, "Greet an 
entity."},
    {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC
initgreet(void)
{
    (void) Py_InitModule("greet", GreetMethods);
}
```

Creditos a: https://gist.github.com/lucasea777/8801440f6b622edd3553c8a7304bf94e

Cython

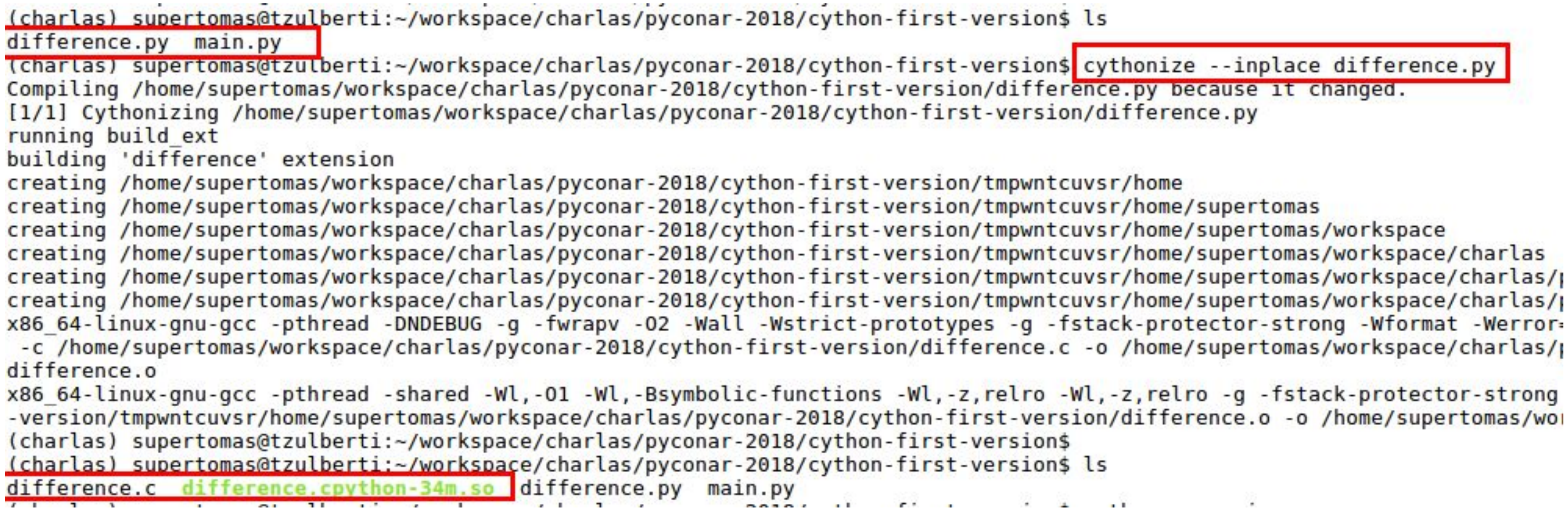
- Permite escribir extensiones de C de Python en Python.
- Es codigo que corre en el Python runtime environment, pero en vez de compilar a bytecode interpretado de Python compila a codigo nativo
- Se instala como cualquier otro paquete de python

```
pip install cython
```

- Seguramente tengan que instalar cosas del sistemas:

```
sudo apt-get install python-dev build-essentials python3-dev
```

Usando Cython



```
(charlas) supertomas@tzulberti:~/workspace/charlas/pyconar-2018/cython-first-version$ ls
difference.py  main.py
(charlas) supertomas@tzulberti:~/workspace/charlas/pyconar-2018/cython-first-version$ cythonize --inplace difference.py
Compiling /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/difference.py because it changed.
[1/1] Cythonizing /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/difference.py
running build_ext
building 'difference' extension
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home/supertomas
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home/supertomas/workspace
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home/supertomas/workspace/charlas
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home/supertomas/workspace/charlas/
creating /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/tmpwntcuvsr/home/supertomas/workspace/charlas/
x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -g -fstack-protector-strong -Wformat -Werror
 -c /home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/difference.c -o /home/supertomas/workspace/charlas/
difference.o
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,relro -g -fstack-protector-strong
-version/tmpwntcuvsr/home/supertomas/workspace/charlas/pyconar-2018/cython-first-version/difference.o -o /home/supertomas/wo
(charlas) supertomas@tzulberti:~/workspace/charlas/pyconar-2018/cython-first-version$
(charlas) supertomas@tzulberti:~/workspace/charlas/pyconar-2018/cython-first-version$ ls
difference.c  difference.cpython-34m.so  difference.py  main.py
```

Archivos Generados por Cython

Cuando se corre **cythonize** comando aparecen dos nuevos archivos

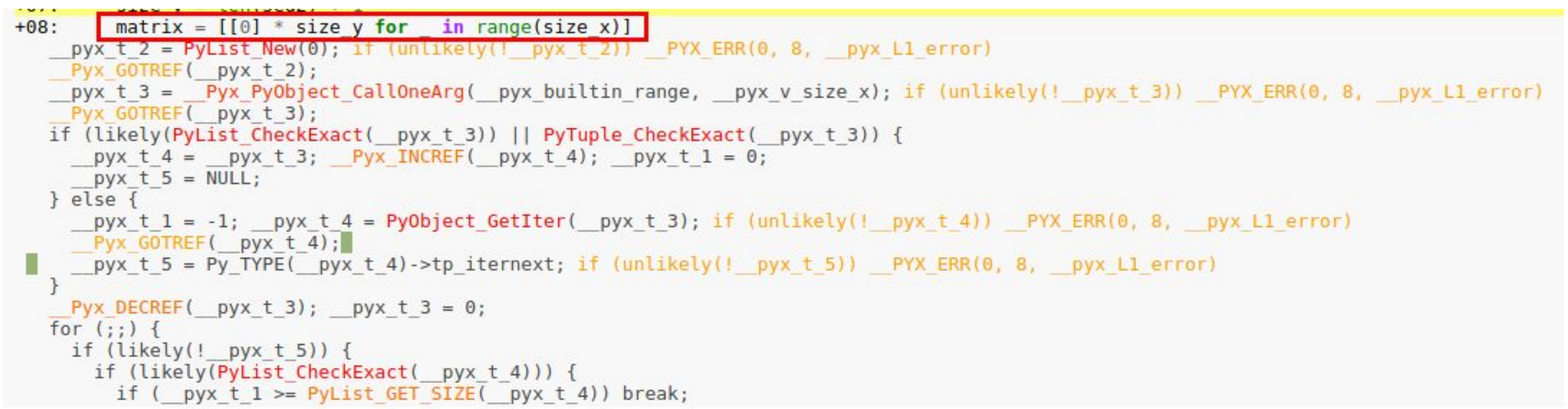
- difference.c es el archivo generado con código C generado a partir del código python generado por Cython

- difference.so o difference.cython.*.so: el sharedlibrary compilado a partir del difference.c

Usando el archivo cythonizado

```
>>> import difference
>>> dir(difference)
['__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', '__test__', 'levenshtein']
>>> difference.__file__
'/charlas/pyconar-2018/cython-first-version/difference.cpython-34m.so'
>>> difference.levenshtein('ola', 'Hola')
1
>>>
```

Codigo generado por Cython



```
+08:     matrix = [[0] * size_y for _ in range(size_x)]
   __pyx_t_2 = PyList_New(0); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 8, __pyx_L1_error)
   __Pyx_GOTREF(__pyx_t_2);
   __pyx_t_3 = __Pyx_PyObject_CallOneArg(__pyx_builtin_range, __pyx_v_size_x); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 8, __pyx_L1_error)
   __Pyx_GOTREF(__pyx_t_3);
   if (likely(PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(__pyx_t_3)) {
     __pyx_t_4 = __pyx_t_3; __Pyx_INCREF(__pyx_t_4); __pyx_t_1 = 0;
     __pyx_t_5 = NULL;
   } else {
     __pyx_t_1 = -1; __pyx_t_4 = PyObject_GetIter(__pyx_t_3); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 8, __pyx_L1_error)
     __Pyx_GOTREF(__pyx_t_4);
     __pyx_t_5 = Py_TYPE(__pyx_t_4)->tp_iternext; if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 8, __pyx_L1_error)
   }
   __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
   for (;;) {
     if (likely(!__pyx_t_5)) {
       if (likely(PyList_CheckExact(__pyx_t_4))) {
         if (__pyx_t_1 >= PyList_GET_SIZE(__pyx_t_4)) break;
```

El archivo en python era de 26 líneas, pero el de C es de 4731 líneas

Comparación de performance

Cantidad de pares de palabras	Python Puro	Cython basico	X veces más rápido
233271	13.846	7.310	1.89
116635	7.785	4.122	1.88
58317	3.506	1.700	2.06
29158	1.561	0.865	1.80

Tenemos una mejora del 80% en performance sin cambiar nada de código y seguimos escribiendo código python, pero sigue siendo **151** veces más lento que C

Ayudando a Cython

Le podemos indicar el tipo de las variables a Cython para que pueda generar código más óptimo. Hay dos formas:

- PXD
- Decoradores

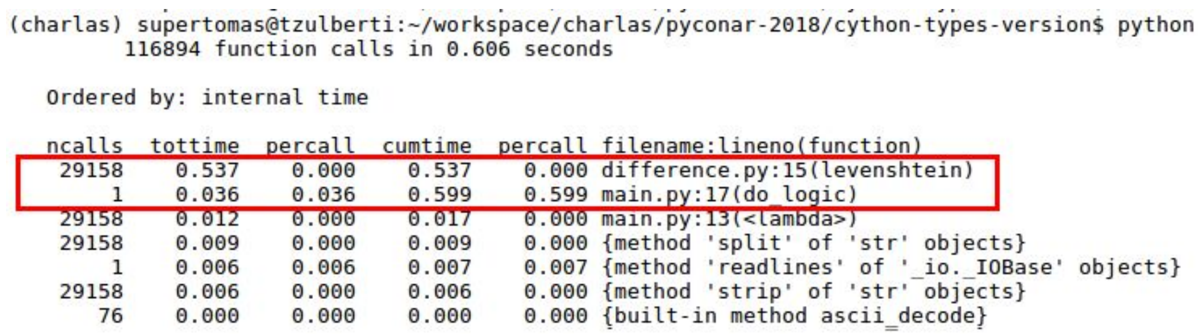
```
import cython

@cython.locals(
    seq1=str,
    seq2=str,
    matrix=list,
    size_x=cython.int,
    size_y=cython.int,
    x=cython.int,
    y=cython.int,
)
def levenshtein(seq1, seq2):
    size_x = len(seq1) + 1
    size_y = len(seq2) + 1
    …
```

Comparación de performance

Cantidad de pares de palabras	Python Puro	Cython con tipos	X veces más rápido
233271	13.846	3.516	3.93
116635	7.785	1.967	3.95
58317	3.506	0.863	4.06
29158	1.561	0.476	3.27

Tenemos una mejora del **400%** en performance sin cambiar nada de código y seguimos escribiendo código python, pero sigue siendo **80** veces más lento que C

Buscando el cuello de botella



```
(charlas) supertomas@tzulberti:~/workspace/charlas/pyconar-2018/cython-types-version$ python -m cProfile -s tottime main.py ../dataset.15.txt
         116894 function calls in 0.606 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    29158    0.537    0.000    0.537    0.000 difference.py:15(levenshtein)
        1    0.036    0.036    0.599    0.599 main.py:17(do_logic)
    29158    0.012    0.000    0.017    0.000 main.py:13(<lambda>)
    29158    0.009    0.000    0.009    0.000 {method 'split' of 'str' objects}
        1    0.006    0.006    0.007    0.007 {method 'readlines' of '_io._IOBase' objects}
    29158    0.006    0.000    0.006    0.000 {method 'strip' of 'str' objects}
       76    0.000    0.000    0.000    0.000 {built-in method ascii_decode}
```

main.py

```python
import sys
from difference import levenshtein

OUTPUT = 0

def main():
    levenshtein_time_taken = 0
    with open(sys.argv[1]) as input_file:
        file_content = input_file.readlines()
        file_content = map(lambda line: line.strip(), file_content)
        do_logic(file_content)

def do_logic(file_content):
    for input_line in file_content:
        string1, string2 = input_line.split(',')
        diff = levenshtein(string1, string2)
        if OUTPUT:
            print('%s %s %d' % (string1, string2, diff))

if __name__ == '__main__':
    main()
```

jampp

Comparación de performance

Cantidad de pares de palabras	Python Puro	Cython all the things	X veces más rápido
233271	13.846	2.535	5.46
116635	7.785	1.288	6.04
58317	3.506	0.665	5.27
29158	1.561	0.355	4.39

Tenemos una mejora del **500%** en performance sin cambiar nada de código y seguimos escribiendo código python, pero sigue siendo 50 veces más lento que C

jampp

jampp

Magia pura

```
@cython.locals(
    cseq1 = 'const char *', cseq2 = 'const char *', pmatrix = 'int[:,:]')
def levenshtein(seq1, seq2):
    size_x = len(seq1) + 1
    size_y = len(seq2) + 1
    pmatrix = matrix = numpy.zeros((size_x, size_y), numpy.int32)

    for x in range(size_x):
        pmatrix[x][0] = x
    for y in range(size_y):
        pmatrix[0][y] = y

    cseq1 = seq1
    cseq2 = seq2

    for x in range(1, size_x):
        for y in range(1, size_y):
            if cseq1[x-1] == cseq2[y-1]:
                substitution_cost = 0
            else:
                substitution_cost = 1
            pmatrix[x][y] = min(pmatrix[x-1][y] + 1,  pmatrix[x][y-1] + 1, pmatrix[x-1][y-1] + substitution_cost)

    return pmatrix[size_x - 1][size_y - 1]
```

Comparación de performance

Cantidad de pares de palabras	Python Puro	Cython con magia	X veces más rápido
233271	13.846	0.723	19.15
116635	7.785	0.438	17.77
58317	3.506	0.249	14.08
29158	1.561	0.165	9.46

Tenemos una mejora del **1511%** en performance sin cambiar nada de código y seguimos escribiendo código python. Aunque sigue siendo **15** veces más lento que C

jampp

LEFT
EXIT 12
ESCRIBIR TODO EL CÓDIGO
BUSCAR EN PYPI
imgflip.com

jampp

Comparación de performance

Cantidad de pares de palabras	Python Puro	Levenshtein PyPi	X veces más rápido
233271	13.846	0.267	51.85
116635	7.785	0.158	49.27
58317	3.506	0.095	36.9
29158	1.561	0.053	29.45

Tenemos una mejora del **4125%** en performance sin cambiar nada de código y seguimos escribiendo código python. Aunque sigue siendo **5** veces más lento que C