

Jampp <> ECI

Python de alta performance



Valentin Paz Marcolla
Software Engineer

Agenda



Cython

- ¿Qué es? ¿Para qué sirve?

Distancia Levenshtein

- Definición y solución
- Pure python
- Pure C
- Cython
- Tipos y Cython
- Chequeo de bordes

De yapa

- Comparación con Pypy (HLL)
- Casos de uso (clases, memory views, GCC functions)

Cython

Performance C en Python?!?!

1

Compilar lo
que se
pueda

2

Interoperar
con CPython

3

Responder
17Kqps por
instancia

Jampp

Cython

Proceso de compilación

```
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
```



Se usa cython para
traducir el código
Python a C

```
__pyx_t_2 = PyList_New(0); if (unlikely(!__pyx_t_2))  
    Pyx_GOTREF(__pyx_t_2);  
__pyx_t_3 = __Pyx_PyObject_CallOneArg(__pyx_builtin_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_5 = NULL;  
} else {  
    __pyx_t_1 = -1; __pyx_t_4 = PyObject_GetAttr(__pyx_t_3, __pyx_t_1);  
    __Pyx_GOTREF(__pyx_t_4);  
    __pyx_t_5 = Py_TYPE(__pyx_t_4)->tp_includes_tuple;  
}
```



Se compila a un .so



El ejecutor de Python
(cpython) linkea las
bibliotecas que tiene
compiladas

00000000	0000	0001	0001
00000010	0000	0016	0000
00000020	0000	0001	0004
00000030	0000	0000	0000

Cython

```
        ibids_per_imp.setdefault(impkey, []).extend(ibids)
__pyx_t_4 = PyList_New(0); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 3743, __pyx_L1)
__Pyx_GOTREF(__pyx_t_4);
__pyx_t_1 = __Pyx_PyDict_SetDefault(__pyx_cur_scope->__pyx_v_ibids_per_imp, __py:
__Pyx_GOTREF(__pyx_t_1);
__Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
__pyx_t_4 = __Pyx_PyObject_GetAttrStr(__pyx_t_1, __pyx_n_s_extend); if (unlikely
__Pyx_GOTREF(__pyx_t_4);
__Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
__pyx_t_1 = NULL;
if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_4))) {
    __pyx_t_1 = PyMethod_GET_SELF(__pyx_t_4);
    if (likely(__pyx_t_1)) {
```

Código original (Python)

Código generado (C)

Resultado del annotate

Sugerencias o instrucciones para Cython

- Tipos
- Nullificable
- Chequeo de bordes
- Objetos const
- Llamada a funciones en C
- Llamada a funciones del compilador
- Manejo de errores
- Interlineado de funciones (inlining)
- C Imports
- Clases cythonizadas (!!!)
- Métodos solo usables desde Cython
- Métodos públicos y privados

Distancia Levenshtein

Distancia Levenshtein

Las operaciones

- **Inserciones:**

Ola => **H**ola

Pesa => **S**pesa => So**p**esa => Sopesar

- **Eliminaciones:**

Hola => **I**ola

Sopesar => S**I**pesar => Spesa**I** => **I**Pesa

- **Sustituciones:**

Todos => Toda**s**

Ingerir => In**j**erir => Inj**u**rir => Injur**i**a

Distancia Levenshtein

La definición

Dadas dos cadenas A y B se define la distancia de Levenshtein ($d(A, B)$) como la mínima cantidad de operaciones que se deben hacer hasta que las cadenas sean iguales. Las operaciones pueden ser cualquiera de las anteriores (inserción, eliminación y sustitución).

1. Todos \Rightarrow Todas } $d(\text{Todos}, \text{Todas}) = 1$

1. AD \Rightarrow ABD
2. ABD \Rightarrow ABCD

} $d(\text{ABCD}, \text{AD}) = 2$

1. ABCD \Rightarrow ABD
2. ABD \Rightarrow AD } $d(\text{ABCD}, \text{AD}) = 2$

1. ABCD \Rightarrow ABD
2. ABD \Rightarrow AD
3. AD \Rightarrow AE

} $d(\text{ABCD}, \text{AE}) = 3$

Distancia Levenshtein

La solución (recursiva)

$|A| = N$ y $|B| = M$

$d(A, B) = d(A[0:N], B[0:M])$

$d(A[0:i], "") = i$

$d("", B[0:j]) = j$

$d(A[0:i], B[0:j]) = \text{distancia del paso anterior } (i - 1 \text{ ó } j - 1)$

+ el costo de hacer alguna operación en éste paso

Distancia Levenshtein

La definicion (recursiva)

si $i = 0 \rightarrow j$

$d(A[0..i], B[0..j]) =$

“” \rightarrow “1” $\rightarrow \dots \rightarrow$ “123...j” j inserciones

“123...j” $\rightarrow \dots \rightarrow \dots \rightarrow$ “123...j” 0 modificaciones

Distancia Levenshtein

La definicion (recursiva)

si $i = 0 \rightarrow j$

si $j = 0 \rightarrow i$

$$d(A[0..i], B[0..j]) =$$

“123...i” $\rightarrow \dots \rightarrow \dots \rightarrow$ “123...i” 0 modificaciones

“” \rightarrow “1” $\rightarrow \dots \rightarrow$ “123...i” i inserciones

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \\ \text{si } i > 0 \wedge j > 0 \rightarrow \end{cases} d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\}$$

Corresponden al mismo lugar en la cadena.
Pueden ser iguales o no.

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \\ \text{si } i > 0 \wedge j > 0 \rightarrow \end{cases} \begin{cases} d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\} \\ d(A[0..i], B[0..j-1]) + 1 \end{cases}$$

$$\begin{array}{ccc} \dots | A[i-1] | A[i] | & \rightarrow & \dots | A[i] | B[j] | \\ \dots | B[j-1] | B[j] | & \rightarrow & \dots | B[j-2] | B[j-1] | B[j] | \end{array}$$

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \\ \text{si } i > 0 \wedge j > 0 \rightarrow \min \begin{cases} d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\} \\ d(A[0..i], B[0..j-1]) + 1 \\ d(A[0..i-1], B[0..j]) + 1 \end{cases} \end{cases}$$

... | $A[i-1]$ | $A[i]$ \rightarrow ... | $A[i-2]$ | $A[i-1]$ | ~~$A[i]$~~

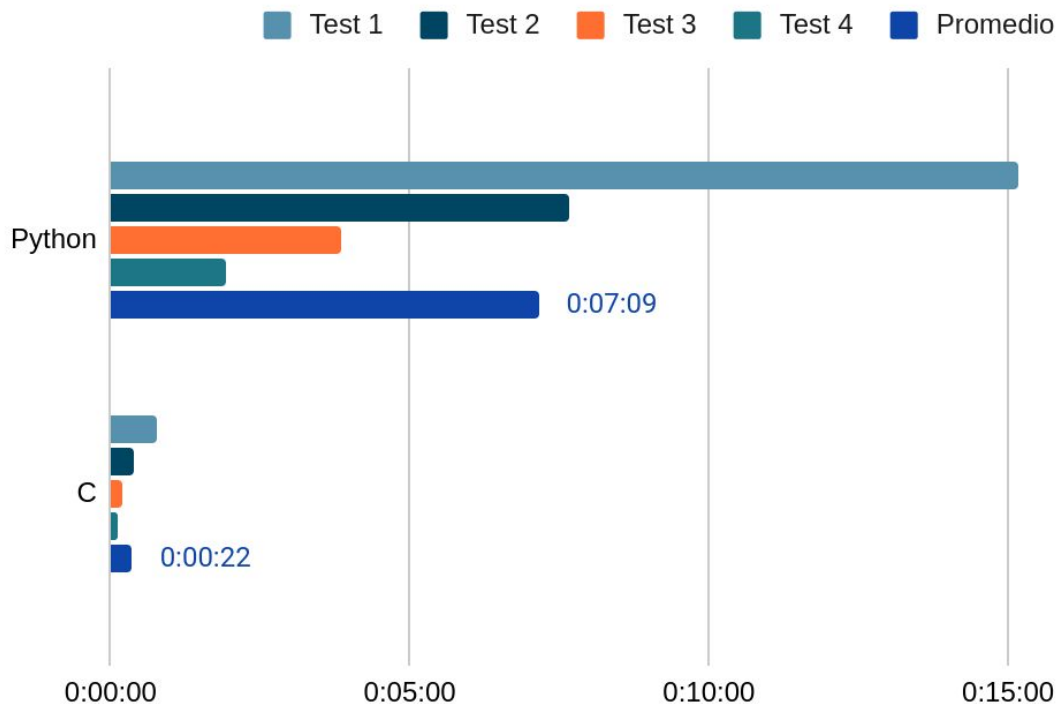
... | $B[j-1]$ | $B[j]$ \rightarrow ... | $B[j-1]$ | $B[j]$

Implementaciones Python y C PD bottom-up

Demo

Resultados Python y C

Minutos por ejecución - Python y C



C es **19x** más
rápido que Python

Hinting Tipos

```
size x = len(seq1) + 1  
t_1 = PyObject_Length(  
t_2 = PyInt_FromSsize_t(  
PyObject_
```

PyInt_FromSsize_t

PyObject

PyNumber

PyList_CheckExact

PyObject_CallOneArg

Son tipos de Python poco específicos,
no nativos. Para hacer indexing,
sumas y restas son lentos.

```
Pyx PyObject_CallOneArg(__pyx_builtin_range, __pyx_  
(__pyx_t_3);  
PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(  
Pyx PyObject_
```

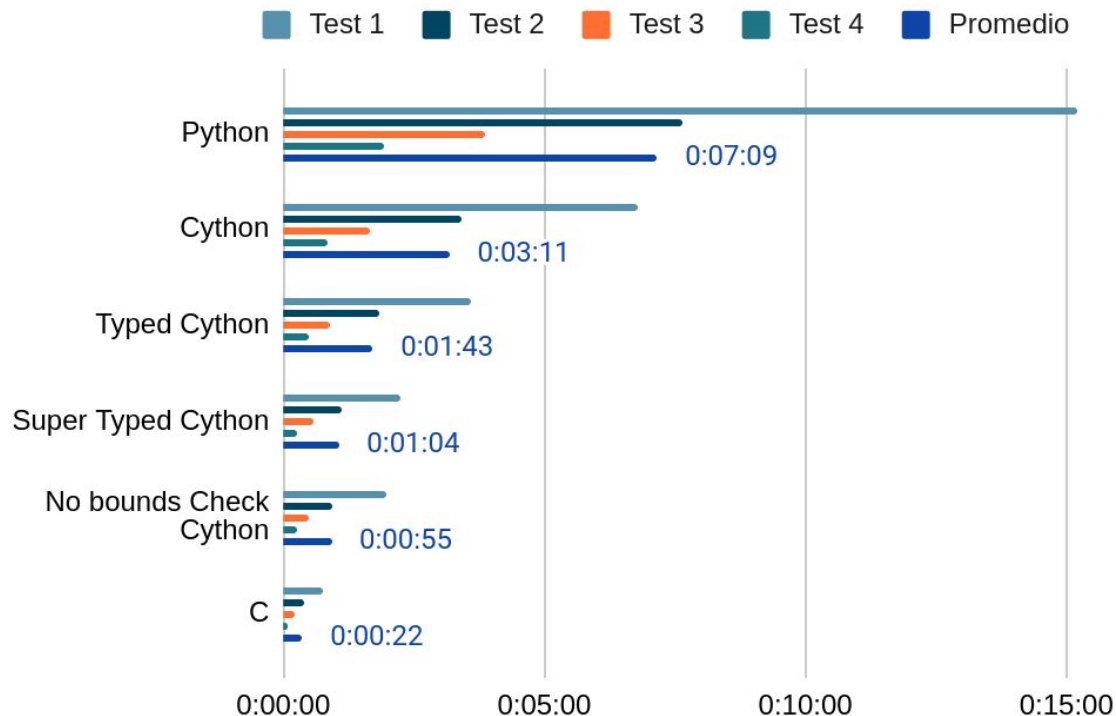
```
PyTuple_GET_ITEM(__pyx_t_4, __pyx_t_1);  
PySequence_ITEM(__pyx_t_4, __pyx_t_1);
```

```
PyObject* __pyx_temp = PyNumber_InPlaceMultiply(  
Pyx PyObject_
```

```
PyNumber_Add(__pyx_t_2, __pyx_  
(__pyx_t_13);  
(__pyx_t_2); __pyx_t_2 = 0;  
(__pyx_t_6); __pyx_t_6 = 0;  
PyObject_RichCompare(__pyx_t_1  
Pyx PyObject_IsTrue(__pyx_t_  
(__pyx_t_14): __pyx_t_14 = 0;
```

Resultados Cython

Minutos por ejecución - Python, Cython y C



Cython es **2.3x** más rápido que Python

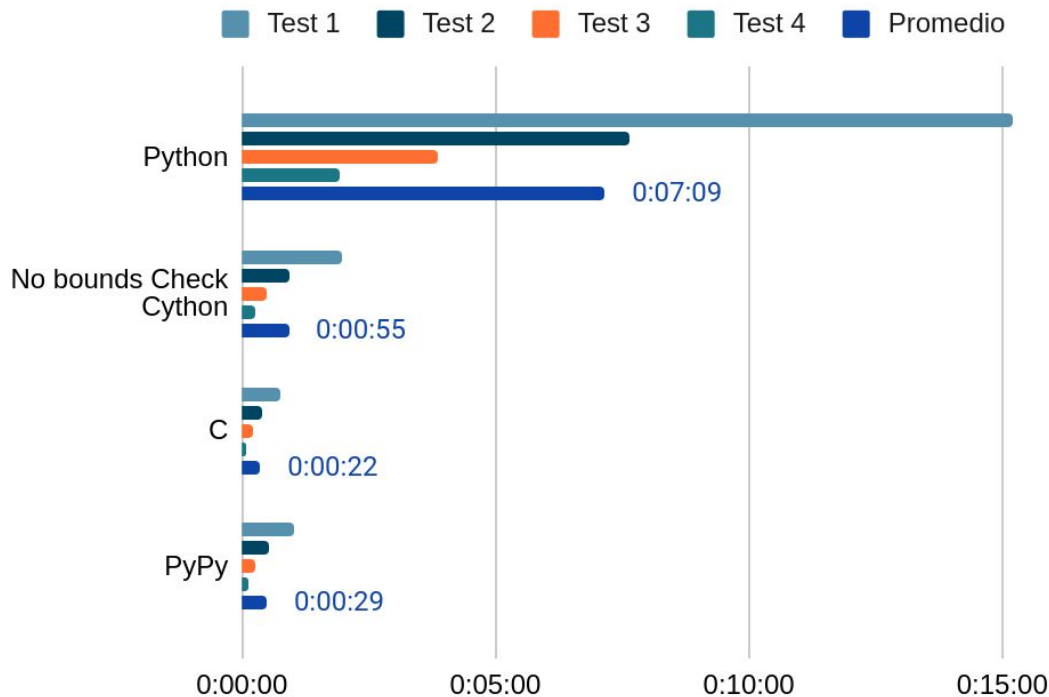
Cython tipado es **85%** más rápido que Cython

La versión final en Cython es **87%** más rápida que Cython tipado.
Y solo **2.5x** más lenta que C.

Con Cython mejoramos **7.8x** la performance de Python.

Contra PyPy

Minutos por ejecución - Python, Cython, C y PyPy



PyPy es un intérprete de Python con compilación **Justo-A-Tiempo** basado en **trazas**.

Es **14.7x** más rápido que Python, **1.8x** que Cython y **31%** más lento que C.

¡Pero ojo!
No funciona bien siempre.

Preguntas?

Cython++

Clases

```
cdef class ConjuntoAproximado:
    cpdef public int seed
    cpdef int _closed
    cdef unsigned char[:] elementos

    cdef inline _insertar(self, int indice, int valor, bint rebalancear)
    cpdef public insertar_elementos(self, elementos)
    cpdef public insertar_elemento(self, elementos, rebalancear=?)
```

Ocupan memoria como si fueran una struct de C.

Usar atributos sin costo natural de Python.

En casos simples: sin conversiones de tipos o comparaciones complejas.

Una llamada a `insertar_elemento` se transforma en una llamada en C, sin costo extra.

Y una llamada a `_insertar` se compila en línea.

```
+318:         if self._closed == 0:
            __pyx_t_1 = ((__pyx_v_self->_closed == 0) != 0);
            if (__pyx_t_1) {
                /* ... */
            }
+319:         self._closed = 1
            __pyx_v_self->_closed = 1;
```

Cython  NumPy

(DataScience)

HyperLogLog

Exprimiendo NumPy

Python: 66s

PyPy es
5X más
lento que
CPython

NumPy no funciona
bien con PyPy

Si las trazas son muy
largas (profundas) con
PyPy se paga una
penalidad demasiado
alta

PyPy: 331s

Cython: 7.4s

9.4X más
rápido
que
CPython

```
@cython.cfunc
@cython.inline
@cython.locals(length=c
@cython.returns(int)
def _compute_value(hash,
    n_hash = uint64(hash)
    value = (n_hash << le
    # should count the nu
    if not value:
        return 65

    if cython.compiled:
        lz = clz(value)
```

```
cdef inline int clz(unsigned long long x):
    return __builtin_clzll(x)
```

Usando funciones de
GCC,
inlining, cfunc y mucho
Numpy.

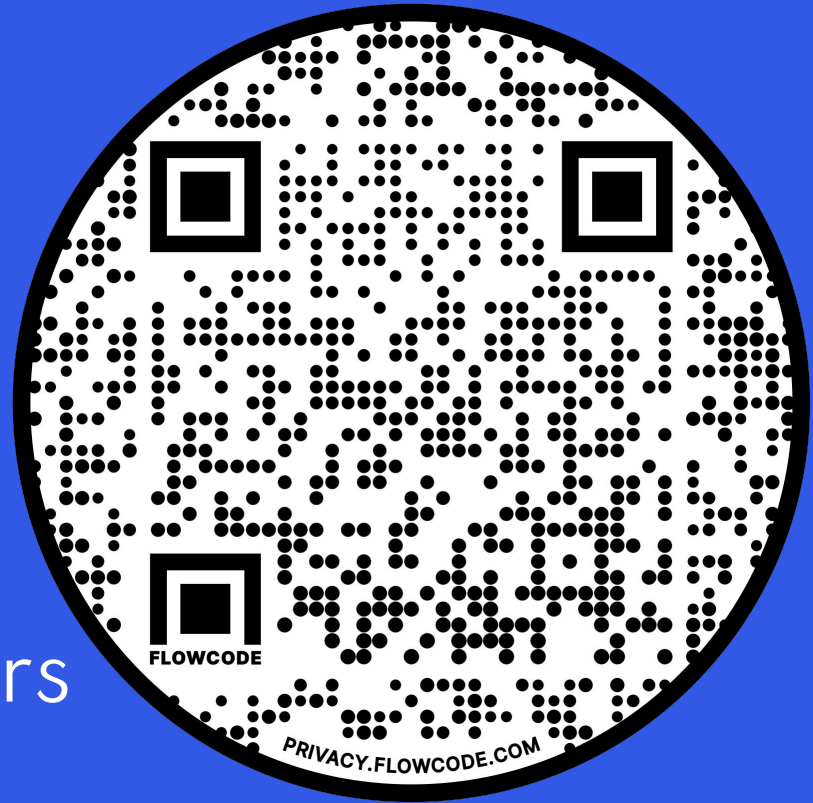
¿Preguntas (bis)?

We are hiring!

www.jampp.com/jobs



Data Analysts
ML Engineers
Performance
Engineers
Data Engineers
Cloud Engineers
Full Stack Engineers
....& more!



Gracias!